TYNE & WEAR

ATARI 8-BIT USER GROUP

Newsletter of TWAUG

Software
Editorial
Buy & Sell
Hardware

TWAUG 8-BIT

Reviews
Help line
Section
Repair Info

Public Domain Library

ISSUE #6

NOVEMBER/DECEMBER 1993

BRING YOUR EIGHT UP TO DATE with power products from COMPUTER SOFTWARE SERVICES

THE BLACK BOX

The BLACK BOX is an add-on board for the Atari 600XL,800XL and 130XE 8-bit computers. It is a T-shaped board that plugs into the PBI port of the XL computer, or the ECI and cartridge ports of the 130XE. Connectors for both types of computers are built into the BLACK BOX so no adapter boards are necessary. A cartridge port is available on the board itself for 130XE users.

The BLACK BOX provides many unique and useful functions. The four primary functions are:-

* RS-232 serial modem port
* Parallel printer port
* SASI/SCSI hard disk port
* Operating System enhancements

The BLACK BOX is $199.95 for the basic unit, and $249.95 with an onboard 64K printer buffer.
Shipping and Handling extra.

THE BLACK BOX ENHANCER

A must for all BLACK BOX owners. The BLACK BOX ENHANCER is a plug-in module for your BLACK BOX, enhancing the printer functions and adding an instantly available, full featured sector editor!

Installation of the BLACK BOX ENHANCER requires one simple solder connection. Only $49.95 plus shipping/handling.

THE FLOPPY BOARD

Our latest and greatest product. The FLOPPY BOARD is an add-on expansion board for the BLACK BOX interface. It allows the use of the same inexpensive floppy drive mechanisms used in IBM computers. The FLOPPY BOARD is the first floppy drive interface to support "high density" floppy drive mechanisms in either 5.25 inch or 3.5 inch. Built into the FLOPPY BOARD are our BLACK BOX ENHANCER and a version of our SUPER ARCHIVER to allow copying of protected disks for 3.5 inch format. Included with the FLOPPY BOARD is our program to read and write to IBM or ST formatted disks. This makes the FLOPPY BOARD the best way to transfer files to and from your 8-bit.

The FLOPPY BOARD is only $149.95 plus shipping & handling.

THE MULTIPLEXER

This device brings the power and flexibility of larger systems to your 8-bit. The Multiplexer is a collection of cartridge interface boards that allow up to 8 Ataris to read and write to the same drives (typically a hard disk), access the same printer(s), and talk to each other. It is the first practical networking system for the Atari 8-bit computer.

One "master" computer (any 8-bit) is equipped with the master Multiplexer interface. Then up to 8 "slave" computers hook up to this master, each having their own slave interface.

The "common" peripherals (things that are to be shared) are connected to the master. On each slave, all disk and printer I/O is routed through the master, so no extra disk drives are needed.

The Multiplexer sells for $199.95 for a master and two slave units with cable. Additional slave units are $89.95 each, plus shipping/handling.

THE SUPER ARCHIVER II

The SUPER ARCHIVER II edits and copies all enhanced density programs plus retains all the features of the SUPER ARCHIVER.

The SUPER ARCHIVER II is only $99.95 plus shipping & handling. NOTICE: if you already have THE SUPER ARCHIVER you may upgrade to S.A.II for only $29.95 plus shipping/handling. Software only.

THE BIT WRITER

The Super Archiver BIT WRITER is capable of duplicating even the "uncopyable" Electronic Arts and Synapse Syn-series, which employ 34 full sector tracks. The BIT WRITER must be used with the SUPER ARCHIVER

The BIT WRITER is only $79.95 plus shipping/handling.

THE ULTRA SPEED PLUS OS

The Operating System that should be in every XL/XE computer! The Ultra Speed Plus puts unbelievable speed and convenience at your fingertips.

Use any DOS to place Ultra Speed formats on your disks (with XF551 or modified 1050 drives), reading and writing at this speed with most programs. This high speed mode can be turned off for maximum compatibility.

Four simple solder connections are required for installation if your machine has a socketed OS ROM. The Ultra Speed OS is only $69.95 plus shipping/handling.

For more information on these and other 8-bit products:

CONTACT
COMPUTER SOFTWARE SERVICES
PO BOX 17660
ROCHESTER, NEW YORK 14617
USA

ORDERING LINE: (716) 429-5639
FAX: (716) 247-7158
BBS: (716) 247-7157

or contact T.W.A.U.G. we will do our best to help.

EDITORIAL

Who to blame!!!

John Matthewson
David Ewens
Max Gerum

I am sorry to say that there has been a little mix up with a documentation disk. I, Max, have been setting up the TextPro documentation files ready for printing and saved all these files onto another disk for the library, unfortunately I gave the wrong disk to David.

We know one or two contributors have purchased the TextPro version 4.56 and received the disk I used to copy from, some of the TextPro files were missing on that disk, could the purchasers please return the documentation disk to TWAUG and we will copy the correct docs onto it.

I am sincerely sorry about this, we wouldn't have known about this mistake if it hadn't been pointed out to us by a friend who purchased the disk set.

I set this doc up using TextPro v.4.56 and also entered some printing codes, it should printout with any Epson compatible printer using the same codes for bold and double height printing. Those are the only codes I've used, the double height print is for the headers only, and emphasized printing I used to highlight some of the paragraph headers. Some of the document sections are larger than one Bank memory but you will find at the end of some files in the first bank a lower case inverse 'g' this will tell the printer to continue printing from bank 2.

The next issue will be ready by mid-January.

CONTENTS

EDITORIAL 3

NEW! PROGRAMMER'S CLUB
Need help programming? 4

HINTS & TIPS
by David Ewens 5

THE PARALLEL BUS REVEALED
by Earl Rice 6

MORE DTP
by Matthew O' Kane 10

FIVE LINERS 14

ADVERT FOR NEW SOFTWARE 15

YOUR OTHER COMPUTER
by John Pickan 16

BASIC TUTORIAL Pt 3 18

PRINTER INTERFACE
Review by RED 20

SOFTWARE TIMERS
by Nir Darey 21

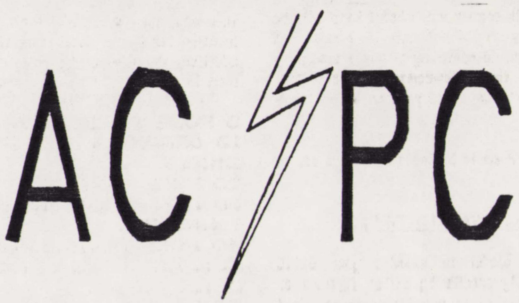
CRACKING THE CODE
by Keith Mayhew 22

MARK'S GAMES COLUMN
by Mark Stinson 28

REVIEWS
by Mark Fenwick 30

DISK CONTENT 31

NEW for the 8-bit Atari...



The Newsletter of
The Atari Classic Programmer's Club

The lack of commercial quality software for the greatest of 8-bits is depressing at the best of times. For too long, we've had to suffer the injustices given to the Atari community by some of the larger software houses, particularly in the UK. So, instead of wringing our hands and watching from the sidelines, why don't you get up and DO SOMETHING ABOUT IT!!

When you join The Atari Classic Programmer's Club, you will be eligible for assistance with any aspect of writing a piece of software on the Atari 8-bit. We can help by providing graphics, Sound, Music, PMGs, UDGs, Play Testing, programmer's helpline, and possible publication of your finished software. We are also happy to help people who have just taken up programming or have just moved to a new language.

We have membership fees to cover 6 months, 12 months and Life memberships, so anybody can join: For full details, send large SAE to:

ACPC, Pen-Tyddyn, Capel Coch, LLangefni, Anglesey, Gwynedd LL77 7UR, Wales.

Everything the Atari programmer needs.

D.G.S. Dean Garraghty's Software.

62 Thomson Ave., Balby, Doncaster. DN4 0NU. TEL. (0302) 855026.

Not only does Dean produce a very good News-Disk, but he also runs an excelent PD library. He has recently branched out into the commercial side of the 8-BIT software scene with the new game called MINE SWEEPER, and more recently the very good new QUICK language.

Why not drop him a line, or give him a phone call and ask him to send you his latest D.G.S catalogue. If you are going to the AMS7 show at Stafford on the 13th of November, why not visit him at his stand and see what he has to offer.

HINTS & TIPS

by David Ewens

Well, here we are again with some more, what I hope will be useful bits and pieces. Although I'm not actually running out of items for this column, I would certainly welcome something from some of you guys out there. Someome somewhere must have a little bit of info that they could pass on to fellow 8-BIT users.

Well, enough chat, let's get down to business and start off at a leisurely pace.

SLOW-MOTION LISTING SCROLLER

Wouldn't it sometimes be useful to examine your BASIC program as the listing slowly scrolls by either forward or backward? That's what you'll get if you insert these eight simple lines of code at the beginning of whatever BASIC program you are working on. Type in the listing below and LIST it to disk. (This program utilizes line numbers 0 to 7, so make sure to start your main program at a higher line number). ENTER the eight-line program from disk after your main program is in memory, and it will be installed at the beginning. Do not use SAVE or LOAD for this program, because that would erase your new program from memory.

Type RUN and you will be prompted for a starting and ending line number. After answering, you may scroll forward or backward one line at a time by pressing either the (SELECT) or (OPTION) keys.

```
0 POKE 710,2:? "START LINE#";:INPUT
L$:? "END LINE#";:INPUT E$:? CHR$(1
25):? :? :?
1 ? "PRESS (SELECT) TO SCROLL FOR
WARD":? "PRESS (OPTION) TO SCROLL
REVERSE":? :? :?
2 LIST L:IF L=0 THEN L=1:LIST L
3 IF L=E THEN END
4 -=PEEK(53279):IF P=7 THEN 3
5 IF P=3 THEN L=L-1:GOTO 2
6 IF P=5 THEN L=L+1:GOTO 2
7 IF P<>3 OR P<>7 THEN 3
```

FIVE COLOUR CURSOR TEXT

ANTIC modes 4 and 5 (same as graphics 12 and 13) are special in that they allow four colours in a single character, and five on the whole screen. However, there is no cursor.

This program sets up a graphics 0 screen and changes to a mixed screen of ANTIC modes 2, 4 and 5. The character Set is altered and the screen colours are changed. Finally text is printed and a cursor is present.

How did this happen? The redefined character set altered the SPACE character, which occupies the whole screen, into a solid block.

This way, the cursor will show up in the colour assigned to location 712 (background) when it overlays the solid block. In addition, the now-solid playfield is it's own colour seperate from the background, so a screen border is possible.

```
5 POKE 106,PEEK(740)-4
10 GRAPHICS 0:DL=PEEK(560)+PEEK(56
1)*256
20 POKE DL+3,68
30 FOR I=DL+6 TO DL+26:POKE POKE
I,4:NEXT I
40 POKE DL+19,5:POKE DL+20,5:POKE
DL+27,65:POKE DL+28,PEEK(560):POKE
DL+29,PEEK(561)
50 REM
100 FOR I=0 TO 1023:POKE PEEK(106)
*256+I,PEEK(57344+I:NEXT I
110 POKE 756,PEEK(106):FOR I=0 TO 7
:POKE PEEK(106)*256+I,255:NEXT I
120 POKE 712,100:POKE 710,148:POKE
711,68:POKE 709,42:POKE 708,0
130 ? "HELLO [ HELLO ]":POSITION 2,
14:"HELLO":POKE DL+22, 2:? :? :? "[
USE ARROW KEYS TO MOVE CURSOR A
ROUND]"
140 RESTORE :FOR I=1 TO 4:READ A:
FOR J=PEEK(106)*256+A*0 TO PEEK(1
06)*256+A*0+7:READ B:POKE J,B:NE
XT J:NEXT I
150 DATA 40,255,107,107,171,107,107,
107,255,37,255,171,191,171,191,191,171,
255,44,255,191,191,191,191,191,171,255
160 DATA 47,255,171,107,107,107,107,
171,255
```

NO QUESTION ?

If you would like an input not to give the annoying "?" try this:

```
100 ? "ENTER NAME:";:input #6,n$
```

Finally, we have had requests for help from people who have had a problem when using Daisy Dot III with Atariwriter Plus. When they have printed out their text file with DD3, they have noticed that it was printing an 'FF' at the top of the text file. The problem is that when you create a text file with Atariwriter, it saves the global format to disk as part of the text file. When you run DD3 and it reads the text file, it tries to print the format lines at the top of the text. The 'FF' is part of this so DD3 prints it out. If you are using Atariwriter with DD3, it is best to change all the options on the global format to ZERO and this will solve the problem.

MORE HINTS AND TIPS

by John Bunting

I can't be the first person to think up some of the following ideas but hopefully some will be new to someone.

Small strips of PVC insulation tape make very good write-protect tabs and a small roll will last a very long time. Perhaps it is safer to avoid red or white in case they allow infra-red to pass.

For disk notching I use a two-hole punch in the following manner. Hold the punch handle depressed and offer a notched disk up to one of the punching shafts. Carefully maintain the disk in this position whilst marking the outline of the disk corner on to the top surface of the punch. I indelibly marked the resultant 'L' shape by scratching it with a small screwdriver. Now I can offer an un-notched disk up to the mark, press the handle and Hey Presto, a neat semi-circular hole in the right place.

I bought my two-hole punch from Stationery Plus for 89p. These and other similar Stationery Supermarkets are in most large towns and are well worth an occasional visit. They often have good offers on suitable material for holding together or binding all those printed-out Docs which we acquire from PD progs. Also I recently picked up 250 1.5 inch labels for 99p. OK so the don't have perforations for the tractor drive but with care they are fine on friction drive particularly on short runs which I usually do.

Whilst on the subject of PD docs; they sometimes come as ASCII files which can be a nuisance to print out. Try using DAISY DOT II (2 not 3) on Epson compatibles or TEN PRINT with Atari 1029s. Both these progs require text in ASCII form and excellent results can be achieved. I recently printed out the docs for Page Editor from TWAUG library using DDII - great.

On ASCII files; don't forget if you edit them you must SAVE them back to the disk in the ordinary way. Don't PRINT to disk which was done originally.

Finally a question: knowing a small amount about BASIC has given me a better understanding of logical procedures in general; if I learned assembler or machine code would I start to understand female logic??

THE PARALLEL BUS REVEALED.

100,000 bytes per second

Part 1 of a four-part series

By Earl Rice.

I was reading through some old magazines when I came across an article called the Parallel Bus Revealed part three of a four-part series. I found it so interesting that I started hunting through the rest of my mags to see if I had the full series. I am pleased to say that I had them all and I have decided to include them in the next four issues of the TWAUG newsletter. Hopefully someone out there will be able to make use of these articles and who knows, may come up with a new add-on for the Atari 8-BIT computer.

David Ewens.

If you own an Atari 600XL or 800XL, you've probably noticed a little plastic cover on the back. Above that cover are the words "PARALLEL BUS." Until now, this port has only been used for memory expansion cartridges.

In June 1984 at the Consumer Electronics Show, the Atari Company finally released full specifications for the Parallel Bus Interface (PBI). This series of articles is based on that information.

In the next few issues of TWAUG, we'll explain how the parallel bus works and how you can use it with your own projects.

IMPORTANCE OF THE PBI

The parallel bus interface runs at the same speed as the 6502 micro-processor -- and it can transfer information more than 40 times faster than the serial connector.

The serial connector can transfer no more than 2400 bytes per second. The parallel bus can easily transfer 100,000 or more per second, depending on software execution speed. This speed allows you to design controllers for hard disks and other high-speed devices.

WHAT THE PBI IS

Basically, the parallel bus connector is an extension of the 6502 data, address, and control signals. These signals aren't buffered, and can drive only a very limited electronical load. Unmodified, there isn't very much you can do with the PBI. When used with appropriate software and hardware, however, the PBI becomes an extremely powerful extension of your computer.

Fortunately, the PBI's design is easy to understand. Additionally, most of the software you'll need is already in the Operating System. This code, called the Generic Parallel Device Handler, resides at location 58511 ($e48f), just waiting to talk to your high-speed device. All you have to do is write the low-level hardware driver software and combine it with your hardware.

THE PARALLEL BUS continued

But first you need to see how the PBI works.

A parallel device (figure 1) is essentially a circuit board containing five key elements.

1: A ROM chip containing both the low-level driver software and a Device Handler Table.

2: Any RAM required for on-board buffers.

3: Some address-decoding logic.

4: A hardware-select register.

5: The functional circuitry itself. (Perhaps an I/O device such as a universal asynchronous receiver/transmitter (UART) to drive a modem, or a parallel interface adapter (PIA) to drive a printer.)

All device registers, ROM, and RAM are mapped in to your computer's memory space as shown in the simplified memory map (FIG 2).

The PBI's ROM space is mapped into the same area as the OS conversion routines from ASCII to Floating Points. The computer's memory management IC switches out the OS ROM when an external device is selected, and switches back in when it's done. The catch is that your external device can't use the Floating Point software in the OS. It also can't use any function of the OS or application software (like BASIC) that uses Floating Point routines.

Since most external devices are essentially I/O peripherals, these restrictions should not create many programming problems.

The First 26 bytes of ROM contain a data table (figure 3). This is a handler table which has the same format as the other OS vector tables. Note that some of the data is optional. The required data consists of ID bytes used by the generic handler to validate the presence of a parallel device, and JUMP vectors to device functions.

During a coldstart, just before attempting to initialize a cartridge, the OS will poll for parallel devices. If the ID bytes are correct, the OS will execute the JMP to the INIT routine at 55321 ($D819) through 55323 ($D81B). This routine must put the address of the genetic handler (58511, or $e48F) into the OS handler table (HATABS) along with the device name (T:, for example).

That Done, your routine sets it's select bit in the Device Mask, performs any device-specific initializations and ends with an RTS instruction.

That's all it really takes to let the OS "talk" to your device. Of course, there are the low-level device drivers to consider, but we'll examine them in a later article. For now, remember that the OS simply needs to know that device exists (have it's bit set in the Device Mask) and to have the generic handler's address in HATABS (figure 4).

The OS can handle up to eight devices on the PBI. The OS selects a device by setting the appropriate bit in the hardware select register, located at 53759 ($diff). BIT 0 selects DEVICE 0, BIT 1 selects DEVICE 1, and so on.

Just like the other registers in the computer, this one has a shadow location. The computer uses shadow registers to update the values in its hardware registers. These values are updated 30 times per second. The hardware select register's shadow location is at 983 ($0247).

SELECTING DEVICES

Before selecting a device, the OS looks at the Device Mask (location 583, $0247) to see if such a device really exists. Recall that this was the bit set by the initialization routine.

Parameters are passed between the OS and the device using the A, X and Y registers plus the Page Zero I/O Control Block (IOCB).

The Carry Flag tells the OS whether or not the device performed its requested function. The device sets the flag when it has performed its function. Otherwise, the carry flag is left RESET (0).

The A register passes a data byte, the X register contains the index to the originating device's IOCB, and the Y register contains a Device Status byte. This is the same as any other Central I/O (CIO) operation.

By the way, this is a good place to mension that Atari's Technical Reference Notes (CO16555 Rev. A) are worth their weight in system errors. The basic operation of CIO, IOCB's, Device Status codes and the like are all presented concisely. If you are serious about writing professional-level software or designing any kind hardware for the Atari computer, this manual is a must. As we go along, I'll briefly explain the concepts you need for these articles, but these explanations are not offered as a substitute for the Tec Reference Notes.

SUMMING UP

So far we've learned: The OS contains a Generic Handler for parallel devices, it selects one of up to eight devices through a hardware register and keeps track of it through a shadow register. The parallel device has a ROM containing low-level driver vectors (and, perhaps, the drivers themselves) and an INIT rutine. During coldstart, the OS will run the INIT routine and the device will declare its existence by writing the bit into the Device Mask and putting its name, along with the Generic Handler's address into HATABS. In operation, the device and the OS communicate through the 6502's A, X, and Y registers plus the Page Zero IOCB. The parallel device cannot use OS Floating Point routines because the device's ROM is mapped into those same locations.

Not too hard, huh? Next issue we'll look at hardware requirements, and after that, we'll work up an example and look at interrupts. In the meantime, try to resist the urge to tear off that little cover. We'll explain how to do it safely in the next issue.

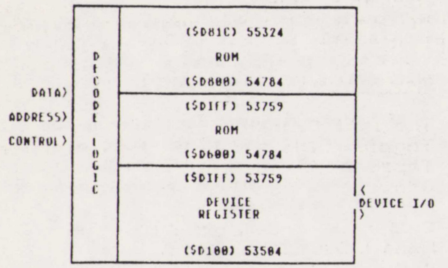


FIGURE 1.
PARALLEL DEVICE BLOCK DIAGRAM

	ADDRESS	OS	DEVICE
58511	$E48F	GENERIC HANDLER	
55324	$D81C	FLOATING POINT ROUTINES	2K ROM
55296	$D800		
55295	$D7FF		RAM
54784	$D600		
53759	$DIFF		SELECT REGISTER
53758	$DIFF		DEVICE REGISTERS
53504	$D100		
584	$0248	SELECT SHADOW	
583	$0247	DEVICE MASK	

FIGURE 2
PARALLEL DEVICE MEMORY MAP

```
$D800  *ROM CHECK SUM LOW
       BYTE
$D801  *ROM CHECK SUM HIGH
       BYTE
$D802  ROM REVISION NUMBER
$D803  ID NUMBER ($80)
$D804  *DEVICE TYPE
$D805  JMP ($4C)
$D806  I/O VECTOR LOW BYTE
$D807  I/O VECTOR HIGH BYTE
$D808  JMP ($4C)
$D809  INTERRUPT VECTOR LOW
       BYTE
$D80A  INTERRUPT VECTOR HIGH
       BYTE
$D80B  ID NUMBER ($91)
$D80C  *DEVICE NAME(ASCII)
$D80D  OPEN VECTOR LOW BYTE-1
$D80E  OPEN VECTOR HIGH BYTE
$D80F  CLOSE VECTOR LOW BYTE-1
$D810  CLOSE VECTOR HIGH BYTE
$D811  GET BYTE VECTOR LOW
       BYTE-1
$D812  GET BYTE VECTOR HIGH
       BYTE
$D813  PUT VECTOR LOW
       BYTE
$D814  PUT BYTE VECTOR HIGH
       BYTE
$D815  STATUS VECTOR LOW BYTE-1
$D816  STATUS VECTR HIGH BYTE
$D817  SPECIAL VECTOR LOW
       BYTE-1
$D818  SPECIAL VECTOR HIGH
       BYTE
$D819  JMP ($4C)
$D81A  INIT VECTOR LOW BYTE
$D81B  INIT VECTOR HIGH BYTE
$D81C  *UNUSED
```

*OPTIONAL -- NOT REQUIRED

FIGURE 3.
ROM VECTOR TABLE

```
$031A  'P' (PRINTER NAME)
$031B  HANDLER LOW BYTE
$031C  HANDLER HIGH BYTE
$031D  'C' (CASSETTE NAME)
$031E  HANDLER LOW BYTE
$031F  HANDLER HIGH BYTE
$0320  'E' (EDITOR NAME)
$0321  HANDLER LOW BYTE
$0322  HANDLER HIGH BYTE
$0323  'S' (SCREEN NAME)
$0324  HANDLER LOW BYTE
$0325  HANDLER HIGH BYTE
$0326  'K' (KEYBOARD NAME)
$0327  HANDLER LOW BYTE
$0328  HANDLER HIGH BYTE
$0329  0    (T)  <------------
$032A  0    ($8F)<------------
$032B  0    ($E4)
$032C  0    (NAME SLOT)
$032D  0    (VECTOR LOW SLOT
$032E  0    VECTOR HIGH SLOT)
$032F  0
$0330  0
$0331  0
$0332  0
$0333  0
$0334  0
$0335  0
$0336  0
$0337  0
$0338  0
$0339  0
$033A  0
$033B  0    (ALWAYS 0)
```

FIND THE FIRST
0 NAME ENTRY, THEN:
PUT DEVICE NAME HERE
PUT GENERIC HANDLER ADDRESS
($E48F) HERE

FIGURE 4.
PUTTING YOUR DEVICE IN HATABS

YORKIE 256K plug-in memory upgrade.

Yes it's available once again, but in limited quantities.

The Yorky simply plugs into the PBI on the back of your 800XL (or internally upgraded to 64k, 600XL) to give you 256k of RAMBO XL compatible bank switched memory. NO soldering is required and you don't have to open up your machine.

It comes complete with a printed manual and a disk full of software designed to make use of the extra memory.

Price 50 pounds + 2 pounds p&p.

For enquiries and orders please write to:-

RICHARD GORE, 79 SPROTBROUGH ROAD, SPROTBROUGH, DONCASTER, DN5 8BW

or telephone me (Richard) on (0302) 784642 any weekend between 7pm Friday and 4pm Sunday.

NB. I can only guarantee this upgrade to work on computers that use a UK standard power supply.

PHOENIX.

The new disk based news letter from Ireland, produced by Robert Paden.

Robert recently had to give up producing the I.A.U newsletter after fur issues because of lack of support.

PHOENIX will be a double sided disk, side 'A' will be packed full of text files containing Articles, reviews and much much more. Side 'B' will contain a good selection of PD software.

The first 6 issues will be available from either the TWAUG, or from New Atari User PD libraries at £2.50 per issue.

If PHOENIX proves to be a success (which we hope it will), then from issue 7, it will only be available from Robert Paden himself.

Issue one of PHOENIX will hopefully be available by the end of August, or sometime in September. Why not give it a try.

MICRO DISCOUNT

Now the largest Mail Order Stockist of Atari 8 Bit Hardware & Software in the U.K.

New Software for 1993

The last Guardian
Tagalon Disks £5.95 H & S Speed Tapes £4.95
Adax Disk £5.95
Eureka Disk £5.95
Guitar Wizard Disk £10.95
Hans Kloss Disk £5.95
and many more

Hardware & Upgrade Kits

Atari Disk drives (re-cond) £100.00
Data Recorders XC12 (re-cond)
Standard £28.00 Turbo £43.00
Upgrades from £18.95 to £51.95
and much, much more

To receive a 32 page catalogue you must first register your name and full postal address on our Data Base. The cost of this service is just £3.60 for 6 copies (1 every 8 weeks): £6.00 Europe: £10.00 Elsewhere. Prices on all items unless stated do not include post and packing, therefore please contact before ordering with details of your requirements to confirm availability and postage. Allow 28 days for delivery.

MICRO DISCOUNT
265 Chester Road
STREETLY
West Midlands B74 3EA
Tel:021-353-5730 or Fax:021-352-1669

NEW ATARI USER PAGE 6

The only magazine left in this country that supports the 8-bit

There is a large Public Domain Library available. Your support is needed to keep the magazine going.

It is now only available by subscription, for more details write to:

PAGE 6
P.O.BOX 54
STAFFORD ST16 1TB

MORE DTP

by MATTHEW O'KANE

Hello again. I admit, once more, that MORE was a little slow in getting out, but now it has arrived. Have a squint at the heading, MORE DTP. We all know the abbreviation of MORE (Matt O'Kane's Review Information) but what is this DTP stuff?

DTP is Desk Top Publishing. A pretty simplified illustration of DTP would have to be PrintShop, where you actually design the page in the computers memory, then send it off to the printer which changes the page in the memory to a page on a piece of paper. So this program will be a DTP type of program.

1 THE NEWSROOM

Then Newsroom pops over the horizon. Newsroom's purpose is to design, edit, and print newsletters and flyers. I understand that it was originally developed to be educational, but don't be scared off by it. Newsroom is very good to publish a sheet so it looks good, and it doesn't take you days to complete it. NR is designed to be ultra-simple to use. I don't know about ultra, but once you grasp the basic swing of it, you're right to handle NR.

The Newsroom program comes in a lovely little hard plastic box. Included inside is the Newsroom Program Disk, the Newsroom Clip Art Disk, a 98 leaf, over-simplified instruction manual; and various pamphlets for special offers, ect. One particular point I couldn't figure out, is that the Newsroom program must boot with BASIC. If you fail to start up with Basic, the program crashes in mid-swing

Plus, a double sided Clip Art disk is included. The Clip Art disk contains over 600 pieces of clip art (so I'm told). Clip Art is like PrintShop Icons, except it is higher resolution. If you don't understand resolution, just think of it in this simple equation.

Resolution = Print Quality

The higher resolution, the better print quality you end up with.

Newsroom is ONLY compatible with the Atari 800XL, 65XE, and 130XE. I have tested it using Revision A BASIC on my 130XE, and it does not work. In addition, I think you need a 1050 disk drive, or compatible. This may rule out the 850 disk drive, so follow this up if you buy it.

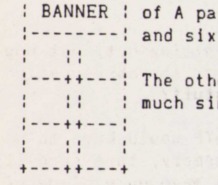
Newsroom works with most 9 pin dot matrix printers, with a "bit image" feature, or if your printer can do graphics. If you are not sure that your printer is supported, just ask me, and I'll look it up. It is NOT compatible with ANY Atari brand printer, EXCEPT the Atari Xmm801.

2 LAYOUT OF A NEWSROOM PAGE

A page designed on NR is made up of banners, panels, and photos. The banner is self-explainatory, its your logo, your slogan, or may advertise top stories in your newsletter. The panels are the areas of the sheet you place text and photos in.

Four "templates" are available for pages. A banner and six panels underneath in two columns, a banner and eight panels in two columns, and two others of eight and ten panels in two columns.

MORE DTP continued



This is a representation of A page with a banner and six panels.

The other sheets are much similar.

3 THE PICTURE MENU

The "picture menu" is just a fancy name for the main menu. It is just a menu with all of the "work areas" drawn in pictures. You can select between the Banner, Photo Lab, Copy Desk, Layout, and the Press work areas.

The menu is the most inivative way of selecting between things I have ever seen! But, like all good things, there is a catch. You must have the Newsroom disk in the drive when you call the menu (unlike the Apple version), and there is a twenty second or so delay while it loads.

Another head banger I bumped into, is that you can jump into DOS, but DOS can't jump back into Newsroom.

4 BANNER WORK AREA

When you click on the Banner icon in the picture menu, you get a considerable delay while it does something to the disk drive.

After this considerable delay, you get a white rectangle - the work area, beside which you see a series of pictures, or icons. You use the Icons to select different features of the Newsroom program.

The first icon in the column is the Clip Art Icon. This is used to grab clip art from the clip art disk. As I said before, provided on a separate disk is a selection from over 600 pieces of clip art of different shapes and sizes. You use the "hand" to pick up the particular piece of clip art you desire, and drop copies of it in the work area. Up to 30 pieces of different clip art can reside in the work area, and you can juggle them around when ever you wish.

The second icon, the flip icon is just to flip clip art from left to right. It is only possible to flip clip art, that is you can't flip something you designed yourself.

Icon Number Three is the crayon icon. When you select this icon, up pops a lovely little menu of all of the graphics tools provided with the program. These tools are VERY limited in features. The user can do the simple stuff, circles, squares, write text, draw lines, fill patters ect., but that's it. In addition, the input devices are limited.

Disk swapping is the biggest hitch. When you call the graphics tools, you must have the NR program disk in Drive 1. Then it takes about 15 seconds to load. This doesn't sound bad, but if you want to keep changing pen thickness, or modes, it is as slow as a snail.

MORE DTP continued

For input devices are the keyboard and the joystick. No koala pads, touch tablets, or light pens permitted in this house. What I mean is that you can only draw with the keyboard and the joystick, which restricts you to clip art.

The keyboard control is lousy. To draw, you have to hold your finger on control, select, and use the arrow keys to move the cursor. Try it now. See how hard it is.

The joystick isn't that much useful either. It either goes too fast or too slow. Clip art takes days to move, but when you are in the magnifying mode, the cursor jumps from one side of the screen to the other in under 0.02 seconds!

There are 5 other icons, pretty much self explainatory. They are the Magnifying Glass, Oops (if you make a boo boo, just click on Oops and it reverses the last command), the Trash Can, the Disk Icon, and the Picture Menu.

One major snag with the disk storage. NR only allows 6 letter file names! Yes, you heard me right! Six letters!

5 PHOTO LAB

The photo lab is essentially the same as the Banner work area, except the design of photos for panels is the main objective. There is one new icon, the camera. When you have finished your photo, click on the camera, draw a frame around your photo, and save it to
disk.

Big drawback: You can not use graphics from other programs (e.g. Atari Artist, Micro Painter, ect.) Also, NR eats disks like its going out of fashion. It demands a separate file name for each panel, photo, banner, and page. If you add it up, you can only get 2 average pages on one disk.

6 THE COPY DESK

Here is the real stuff. This is where the text is actually written. Before your eyes is a couple of new icons, the FONT icon, for selecting different type styles for text, and the Eraser for wiping all of the text in the panel.

The font icon is plainly the same deal as the Graphics Tools, except you only choose the type font you would like. The disk whirls as a little menu loads with all of the type styles on it. Inconveniently, the slow loading speed and disk swapping is the same as the Graphics Tools.

It is claimed on the package that there is an advanced Word Processor included in the program. Baloney! You can't load text from other word processors, no underline, justification, auto centering, ect. All the things you miss.

If you are a fast typist (like me) you will have to slow down. Right after you trash a panel, and start typing a new one, nothing happens for a couple of days, then some of the letters appear two months later, out of order.

Deleting is slow too. You can't appreciate how damn slow it is. It annoys you when you start typing, and nothing happens for a while, or it skips letters you typed.

MORE DTP continued

7 LAYOUT?

Then we arrive at the layout area, with a bang. This one is used for setting out the final page. It works. That's it. Well, most if it works. It does say in the manual that you can copy disks in the layout area, but when you do, it reads in the file to copy, then writes it back to the same disk. Great guys!

Also, there is only 4 possible layouts you can have. They could have allowed a bit of room, I think.

8 THE PRESS

Pretty self explanatory. This is where the printing is done. Draw backs are, no auto copy feature for those times when you want to make ten or so copies.

9 CONCLUSION

I know I have raved on a little, but you must know the full picture before jumping into this program. It is $99 dollars at Computer-1 these days. Newsroom has its good points and its bad points. Its up to you whether the good outweigh the bad.

See you again in MORE.

FIVE LINER

ELECTRONIC TYPEWRITER

from G.D.Massey

This five liner was first published in the Old Atari User.

This Five-liner allows your Atari keyboard to act like an electronic typewriter using its normal printer.

The screen acts just like the old memo-pad mode on the old 400/800 Atari computers, except for the fact that the control codes go straight go straight to the printer rather than doing their normal function on the screen. Thus you can turn printer options on and off by sending the special codes. Some people may wish to add a POKE 752,1 to remove the cursor.

```
10 PRINT CHR$(125):OPEN #1,4,0,"K:"
20 OPEN #2,8,0,"P:"
30 OPEN #3,8,0,"S:"
40 FOR G=0 TO 1 STEP 0:GET #1,K
50 PUT #2,K:PUT #3,K:GOTO 40
```

FIVE LINERS

These FIVE - LINERS were first published in the old Atari User.

THE BLACK LINE way to more legible screens

by Len Golding

This routine works by changing the display list, naturally enough, but it has to cheat a bit because there's a potential conflict between the Antic chip and the screen editor.

The editor assumes that there are always 24 text lines in a Graphics 0 screen, and it won't notice the thin black ones. But Antic controls the TV display and it wants to add extra lines,which means your TV display will get taller.

Twenty three black lines are needed, and each is one-eigth of a Graphic 0 line high. So we need to accommodate the equivalent of three extra text lines on screen.

Fortunately Atari left very generous top and bottom margins when they designed the hardware, so most TV's can cope provided they are adjusted correctly.

The new display list sits in Page 6 - where else? - and the data statements show what it looks like. Every 0 produces a blank line, and every 2 a normal text line.

The first three numbers added together determine the top margin, and you will probably need to experiment with other values to suit your own TV.

Legal numbers are 0, 16, 32, 48, 64, 80, 96 and 112. Dont forget that this is bending the rules, as all display lists should, according to Atari, start with three 112's.

A display list is not a true machine code routine, although it looks a bit like one. It's more of a data table, which Antic refers to as it draws the TV display.

You can therefore change some of the numbers without locking everything up - though you will get some peculiar results if you alter any but the first three.

Lines 10 to 30 copy the new display list into Page 6, starting at address 1536. Line 40 tells this display list where the screen is located in memory, and Line 50 gives Antic the new address to tell it to start using the new display instead of the standard one.

Once this has happened the Basic loader routine is no longer needed. Type NEW and the new display will still remain.

Pressing System reset will bring back the conventional screen, but the ruled lines can be restored by repeating the POKE instructions at Line 40.

```
10 FOR X=0 TO 54:READ D:POKE 1536+X,D:
NEXT X
20 DATA 16,16,80,66,255,255,0,2,0,2,0,
2,0,2,0,2,0,2,0,2,0,2,0,2,0,2
30 DATA 0,2,0,2,0,2,0,2,0,2,0,2,0,2,0,
2,0,2,0,2,0,2,0,2,0,2,65,0,6
40 POKE 1536+4,PEEK(88):POKE 1536+5,PE
EK(89)
50 POKE 560,0:POKE 561,6
```

DICE

by Len Golding

This program simulates a die, displaying one of six possible faces after every roll..

Packing everything into five lines is tricky. You can't simply generate random dots on the screen because only six of the possible combinations look like real dice faces.

This five liner gets round the problem by storing the six legal patterns in an array then using a random number (0 to 5) as an index into it.

A die face has three columns and three rows, so we need nine bytes of information to store each legal pattern of dots and spaces. Six possible faces therefore require a total of 54 bytes.

There's no need to arrange them as 3 x 3 matrices in the computers memory - a 54 x 1 array will do just as well and can be handled just as easily.

Line 10 switches the cursor off, clears the screen, sets up the array and puts the number 32 - blank space character - into every element. Then it opens the keyboard for input.

Line 20 holds the positions, (4th, 9th, 17th, 20th and so on) where dot characters - CHR$(20) - are to be stored. If you drew the array out on graph paper you would see that each block of nine entries can be split into three sets of three, then arranged one under the other to form a die face.

Line 30 does nothing more than to draw a box shape to contain the dots.

The POKE 764,1 command is there to fool the computer into thinking you've pressed a key. Without it your initial screen would just contain a erather enigmatic box with no dots in it.

Line 40 is where the main loop starts. It waits for any character key to be pressed, then generates two ramdom numbers. The first is between 5 and 15 and determines how many times the die will roll before it settles.

The second ranges from 0 to 5 and determines which face will finally be displayed. This second number is multiplied by nine to find whereabouts in the array the chosen fsce pattern starts. Line 40 also produces a beeping sound to accompany the pattern changes.

Line 50 retrieves the chosen nine bytes, chops them into three sets of three and prints each character - dot or space at its correct position inside the box on screen.

```
10 POKE 752,1:? "K":DIM A(53):FOR X=0
TO 53:A(X)=32:NEXT X:FOR X=0 TO 20:REA
D D:A(D)=20:NEXT X:OPEN #1,4,0,"K:"
20 DATA 4,9,17,20,22,24,27,29,33,35,36
,38,40,42,44,45,47,48,50,51,53
30 POSITION 17,9:? "┌──┐":FOR X=10 TO
 12:POSITION 17,X:? "|  |":NEXT X:POS
ITION 17,13:? "└──┘":POKE 764,1
40 POSITION 13,22:? "PRESS ANY KEY":GE
T #1,A:FOR T=1 TO INT(RND(0)*10)+5:N=I
NT(RND(0)*6)*9:SOUND 0,T*3+N+10,10,8
50 FOR R=0 TO 2:FOR C=0 TO 2:POSITION
C+18,R+10:PRINT CHR$(A(N+R*3+C)):NEXT
C:NEXT R:NEXT T:SOUND 0,0,0,0:GOTO 40
```

FIVE LINER

This five liner was first published in the old Atari User

DRUM BALL

from Ivan Mackintosh

Here is a simple Basic demo which bounces a little ball around the screen. It uses a modified display list and drum-type sound effects, both of which could be adapted for use in your own programs.

Line 10 sets up the program variables.
Line 20 sets up the modified display list, and prints out the title line.
Line 30 prints the ball - a (Control T) character - and checks the X coordinate.
Line 40 checks the Y coordinate of the ball.
Line 50 makes the drum sound effect. Then it removes the ball from its old X, Y position, increments X and Y accordingly and loops back to line 30 to print the new ball and go through the loop again.

```
10 GRAPHICS 0:X=10:Y=10:OX=X:OY=Y:DX=1
:DY=1:POKE 752,1:D=PEEK(560)+256*PEEK(
561):POKE D+3,71:POSITION 6,0
20 ? "DRUMBALL":POKE D+6,6:FOR T=18 TO
 28:POKE D+T,6:NEXT T
30 POSITION X,Y:? "●":IF X<1 OR X>38 T
HEN DX=-DX:V=8
40 IF Y<2 OR Y>10 THEN DY=-DY:V=8
50 V=V-(V>0)*2:SOUND 0,X,8,V:X=X+DX:Y=
Y+DY:POSITION OX,OY:? " ":OX=X:OY=Y:GO
TO 30
```

ATARI CLASSICS

The Magazine for the Dedicated 8-Bit User

ENERGIZE YOUR 8-BIT: Plug into AC!
ATARI CLASSICS, that is!
SUBSCRIBE NOW !!
Rates: Foreign (All) 3rd Class Mail
$32 for 1 year
Europe/Mediterranean
AIRMAIL $38 for 1 year
Payment by I.O.M or VISA - MASTERCARD
(Credit Card orders $2 processing fee under $100)

ATARI CLASSICS, 179 Sproul Road/RT.352, Frazer, PA 19355-1958 or Phone 313-973-8825.

ULTIFONT

ULTIFONT has been written exclusively for T.W.A.U.G... and it is only available from our library, for £4.95.

It is the most powerful character set editor written for the Atari. It fully supports the Atari 4-colour text modes and allows you to edit two adjacent characters as one. In addition, you may edit 2 different fonts at once and easily work between them.

It is totally joystick-driven, so you can sit back and relax, rather than hunching over the keyboard. ULTIFONT requires a colour display and 48K of memory.

Printed instructions come with the single sided disk.

TYNE & WEAR ATARI USER GROUP

T.W.A.U.G.
P.O.BOX No.8
WALLSEND
TYNE & WEAR
NE28 6DQ

YOUR OTHER COMPUTER

by John Picken GCACE
from PSAN April 1990

(Reprinted with permission and with THANKS by the OL'HACKERS ATARI U.G.Inc., Oceanside, NY

I am sure a lot of readers will find this article very helpful.
I would like to thank John Picken for writing this article and also thank OHAUG for giving us permission to publish it.

Do you neglect your other computer the one at the end of the printer cable? Today's low cost, dot matrix printers contain at least one microprocessor, a ROM with multiple character sets, and sufficient RAM for buffers and user designed characters. This makes them computers -- in fact it makes them pretty powerful text processors. The majority of owners, however, rarely do anything with their printers that can't be done as part of some pre-packaged software.

A few years ago, you couldn't gain much by learning to talk directly to your printer -- you could get commercial or PD software to exploit nearly all of its features. But improvements in printer ROM's have given the latest models capabilities that were unknown when programs such as "The PrintShop" and even some of the most recently released word processors were authored. If you want to access these features, there's no way around the necessity of learning how to talk to your printer.

Talking to your printer is not a terribly complex task. Unfortunately, however, many printer manuals appear to have been written by the same people who wrote the "manuals" packaged with the 800XL. While it is possible to puzzle through them, assuming you have some knowledge of BASIC (MicroSoft variety), it isn't easy. To make matters worse, I'm not sure that the authors of some of these manuals have much knowledge of BASIC (ANY variety).

Assuming you want to learn to control your printer, what can you do about it? First, read all/anything you can lay hands on concerning printers. My printer is a Roland but I have picked up tips from manuals for both Apple and Star printers. The "Epson Printer User's Handbook", published by Ballantine Books with a cover price of $9.95 (US) is a good place to start, but it's only a start: the Raven (and its Panasonic equivalent, the KX-P1180) has many features not mentionned in the book and lacks a few that are.

Don't forget computer magazines. It doesn't imply disloyalty to your favourite brand to read magazines written for other makes of computer. Regardless of computer brand, the odds are that the printers referred to can be and are used with Atari systems.

There are two other important ways to learn how to talk to your printer. The first is to ask questions -- that's what user groups are for! The other method, one you will have to use at some point, is experimentation. Flash up the printer, sit down with your printer manual and start trying out ideas.

Here's a few hints to get you started or to stimulate more ideas. All are based on my experience with various Roland and Panasonic printers. They should work on most Epson compatible units depending on the printer features included.

HINT #1: Read and re-read the manual to ensure you know all the implications of each command you wish to use: Some commands cause loss of the data in the print buffer (eg. a margin change), others effect a carriage return, some change page formatting settings (tabs, etc.), and some may even be ignored depending on previous commands. The only way to learn to make effective use of the various commands is to try them out in different combinations and sequences.

HINT #2: None of the sample programs in the Roland manual work with 8-bit machines -- even after you change the string definitions to Atari format. But, if you OPEN a channel and replace the LPRINT's with PRINT #, the samples work correctly. The lesson here is: learn enough BASIC to be able to send output to the printer without use of LPRINT. Depending on the computer and the BASIC used, LPRINT produces results ranging anywhere from what you want to the totally unpredictable. From what I've read, it appears this is more important with 8-bit's than it is with ST's and up.

HINT #3: Leave your printer's DIP switches in the "as shipped" condition and leave the exterior switches in the "Standard" or "Program" mode -- you get better and more precise control through software. With 8-bit computers, this advice also applies to any switches on the printer interface (another manual to read -- groan).

HINT #4: If you're using a word processor and not getting the results you want, check the program documentation to ensure it's capable of doing what you're asking and that you're asking correctly. (Proper commands and a proper printer "driver" or "CNF" file.) Then try the "Print to disk" option. This allows you to examine the actual output going to the printer using a sector editor, a screen dump of the file, etc. On the 8-bit, a good way to use this technique is with TextPro: you "print" to a disk or RAMdisk file and then load the new file back into TextPro.

HINT #5: Experimenting can take a lot of paper, especially if you're playing with page formatting. Cheap paper is not necessarily a bargain here. Lightweight paper is less expensive, but you can normally only use one side. The extra few dollars spent on 20 lb. paper are worth it when you consider that you can usually use both sides, if not for printing, then at least, as "scratch" paper. Also, you can save a lot of paper (or a lot of rewinding) by ending a file with an "Off Line" or "Remote Deselect" command (19). This causes the printer to quit immediately -- before a word processor can issue a "Form Feed". You'll then have to issue an "On Line" (17) command or re-boot the printer before its next use.

GRAPHICS: Without getting into "Bit Image Graphics", it is possible, with Roland and Panasonic printers, to use character graphics.

SECOND COMPUTER continued

The Raven allows you to switch between Italic and Graphic character sets on the fly. In fact, it is possible to use both at once: you enable the Graphic set and get your graphics by sending bytes greater than 127 while you can still obtain italics using the ESC 4 sequence. With older models, you might have to set the DIP switches and put the printer into "IBM" mode. This entails writing a new printer driver for your word processor and sacrificing the use of italics, but the graphics can really dress up the output.

This is an area where you have to be familiar with documentation. For example, PaperClip will not allow you to enter inverse (greater than 127) bytes, so how do you include graphics? Simple, the printer has commands to force the most significant bit (msb) either high or low and another to tell it to receive all bytes as sent. With PaperClip, set one of the four "user definable" commands in the CNF to enable graphics and set the MSB high. From then on, all normal bytes will print as graphics until you use a second "user definable" command to receive the bytes as sent.

TEXT PROCESSING: Ever dump a text file to the printer direct from DOS or from a terminal program? If you did, you probably weren't overly ecstatic with the output: margins only 1/4" wide, words broken at the end of lines, etc. If you want better results, you usually end up loading the file into a word processor and formatting it. How would you like to be able to dump files directly to the printer and have the results come out already formatted with word wrap and proper margins on all four sides?

It's easy if your printer includes commands for perforation skip and auto justification. The following program produces a short file that can be sent to the printer prior to a dump. RUN it once and save the file it produces. To use it, set the print head at the perforation, then COPY PRINTER.DAT to the printer before you send it a text file. Your printer will output right justified text with 1" margins all around (the printer handles the pagination), with word wrap, and in highly readable, proportional text. The program is in Atari BASIC, but the DATA statements can be just as easily used with any variety of BASIC, 8-bit or ST.

```
10 OPEN #1,8,0,"D1:PRINTER.DAT"
20 TRAP 60          Just loop 'til
30 READ BYTE        out of data
40 PUT #1,BYTE      error occurs
50 GOTO 30
60 END
70 REM --------------------------
100 DATA 17,13,13,13,13
110 DATA 27,64,27,77
120 DATA 27,108,9,27,81,87
130 DATA 27,112,1
140 DATA 27,97,3
150 DATA 27,67,66
160 DATA 27,78,12
170 DATA 27,107,0
180 DATA 27,120,0
```

Here's the explanation for the bytes in the DATA lines:

100 - We start with an "On Line" command (just in case) and four carriage returns (if the 13's don't work with your 8-bit interface, change them to 155's).

110 - Next we issue a "Reset" just to clear any previous settings and select "Elite" or 12 cpi. (Makes it easier to calculate the 3/4" needed below).

120 - Now, set left and right margins at 3/4" each. With standard paper, this produces actual margins of 1".

130 - We enable Proportional print (this overrides Elite but doesn't change the margins).

140 - Now we select right-justified output which ensures word wrap and microspacing. Note that this command always produces a carriage return.

150 - Designate page length of 66 lines. This is not done for length since 66 is the default, but it is needed to reset the printer's top of page indicator. Reset would also do this but would cancel our previous commands.

160 - Here we enable a 12 line (2") skip perforation which produces the 1" vertical margins.

170 - This command selects one of four NLQ fonts. The final byte in this line can be 0 (default), 1, 3, or 5. If your printer only has a single NLQ font, just omit this line. (The TRAP technique ensures we don't have to count bytes).

180 - The final command selects NLQ. The last byte can be 0 for "OFF" or 1 to enable it. Optionally just skip the line.

It's easy to modify the left and right margins -- just remember that, with standard paper, you always have 1/4" on either side to start no matter what you do. But what about the top and bottom margins? The trick is to get the print head down to the first line we want to print on. Then RESET THE TOP OF PAGE INDICATOR BEFORE DESIGNATING A SKIP PERFORATION VALUE. The skip value must be equal to twice the vertical margins. So for 2" margins, we'd have to add six carriage returns and designate a skip perforation value of 24 lines (4"). It's easier than it sounds; play with it.

CONCLUSION: Since I've begun experimenting with the various printer features, I've found I get better output than I can with any word processor. The techniques outlined above can be most effectively used with a program such as TextPro which allows you to insert and edit printer commands directly in the text. This means you can't use a word processor to "Print" since it would confuse the commands with printable characters; instead you "Save" to a file and then "Copy" it to the printer or even "Save" directly to the printer. I think if you try some of the ideas here you'll be pleasantly surprised -- the operative word is "try".

++*+ end +*+*+*

BASIC TUTORIAL PART 3

Hello dear friends. This is part 3 of our Basic Tutorial and this time I'm going to talk about Basic Languages. Someone around the world will probably curse hard and ask himself what have I been doing till now, if not talking about the Basic Language. Today the article will be about Basic Languages written by other people, which enhances Atari Basic. The article we'll discuss is mainly Turbo-Basic, which I have mentioned in the past, and which I think is the best Basic around for the Atari. I will also discuss in general basic A+.
I don't presume to know all the Basic Languages ever written for the Atari and I have no intention of covering all those that I know, because many of them have grave compatability problems with Atari Basic, and some are not so enhanced as to be worthy of discussion.
As I promised lets discuss Turbo-Basic, which from now on will be referred to as "TB". The first advantage that "TB" has is the compatibility to Atari Basic, which means that a program written in Atari Basic will usually load and run with no problems on "TB". That sounds wonderful but it isn't as simple as it sounds because another big advantage of "TB" is that it runs about 3 times faster than Atari Basic so if you have a program with some delay loops or other time considerations which are important, they will have to be changed. Many of you will now start to wonder if it's worth the trouble, but believe me it is definitely worth while, because everything related to Graphics will draw faster, thus causing us to wait less, In addition because of extended commands (which will be discussed later) "TB" enables the use of less Machine-Language in Basic Programs and thus less initialising and less trouble of understanding the process of using Machine-Language in a Basic Program. Due to extended commands it is possible to write an orderly constructed program in the same fashion as Pascal and other higher Languages.

Until now we have discussed only generally the great benefits of "TB" but lets be more specific. Now, I'm going to give a full and comprehensive list of all the additional commands, their purpose, and their utilization. Now we will see the real power and versatility of this great Basic.

1) *L = "TB" creates indentations according to the conditional structure of the program with the outer most loop without any indentation and the inner most loop with indentions according to the number of loops. This sounds very complicated and very hard to understand but lets make it simple with an example:

```
10 F.A=1 TO 10
20  ? NUMBER
30  F.B=1 TO 10
40   ? NUMBER2
50   ? A
60   ? B
70  N.B
80 N.A
```

This is when you type in direct mode or on the program '*L', but if you don't, it will look as normal Atari Basic without any spaces.

2) *L- = This cancels No. 1

3) TRACE = This a very useful command. This command causes the computer to type the line number it is executing when running and thus the tracing and correction of errors becomes much easier. The command has one disadvantage - when a Graphics call is made it disables itself automatically.

4) TRACE- = Cancels the trace mode.

5) RENUM A,B,C = We have reached heaven because this enables you to renumber your entire program using the following parameters: A - Starting line, B - New starting line, C - Increment. This command is one of the best renumbering utilities for the machine, but it has some disadvantages that to the best of my knowledge no programmer has ever managed to solve: It cannot renumber the folowing lines:

```
10 A=100
20 GOTO A
```

This sequence will not be handled properly, but as you will see later on, you don't need this kind of bad programming.
Another disadvantage is that if a program has lines from 10 to 1000 with an increment of 10 and you renumber lines 100-109 to be 0 till 9 you will find them at their former place but with new line numbers and this will cause grave problems.

6) DEL A,B = Again, one of the more useful commands of "TB". It deletes all the lines between A and B inclusive.

7) DUMP = This command gives you a list of all the program variables with their type, usage, or line number for procedures and labels (which will be discussed later on).
If you type 'DUMP "P:"', you will get a printout of the previous.

8) *B = This small useful command traps the break key. You put at the line you want the computer to jump to when the break key is pressed and it will jump there.

9) *B- = Disables No. 8.

10) ERR = Who among all you Atarians knows how to get the last error number without searching the right 'PEEK' in a book ? Well it is no longer needed. All you do is type:

? ERR - And you will get the last error number.

11) ERL = Same as above, just that this gives the line the last error was found in. (Same as '? PEEK(186)+256*PEEK(187) in Atari basic)

12) -- = This is very useful for structured programing. If you type:

10 -- When you list the program you get:

```
10 --------------------------------
```

BASIC3 TUTORIAL continued

This is a very useful divider between different parts of the program, its much neater and more comfortable to find the parts you need.

13) DPEEK = Same as '? PEEK(A) + 256 * (A+1)'.

14) DPOKE = Same as 'POKE A + 256 * (A+1),B

15) A$=HEX$(A) = A$ holds the hexadecimal equivalent of A. (Dec to Hex conversion).

16) A=DEC(A$) = Same as before just the other way around - Hex to Dec conversion.

17) MOVE A,B,C = Moves a portion of memory from A to B with the size of C. This is very useful because it enables moving players and missiles without machine-language and much more (relocating character set and much much more). If you use - MOVE A,B,C then it will move correctly also overlapping areas of the source and target addresses.

18) POKE $600,X = Almost each computable number can be given to the computer both as a decimal number, and a hexadecimal preceded with the dollar sign.

19) A DIV B = Gives you the full number of times B is contained in A.

20) A MOD B = Gives you the remainder of the division with A and B. (This is very useful to create a repeating loop)

21) TRUNC = True fraction cutting.

? INT(-5.7) = -6
? TRUNC(-5.7)= -5

22) ? 5 & 7 = Boolean and between the bits.

23) ? 5 ! 7 = Boolean or between the bits.

24) ? 5 EXOR 7 = Boolean exclusive or between the bits.

I havent bothered much to explain 22-24 because I haven't found it very useful, but if someone doesn't understand or wants more info, just drop me a line.

25) RND = Gives random numbers between 0 and 1. (Parallel to RND(0) in Atari basic)

26) RAND(N) = Gives random values between 0 and N.

27) X0,X1,X2,#3 = Special pre-defined numbers which save memory. A regular number takes 8 bytes of memory while this foursome requires only 3. (Not very useful, unless there is memory consideration).

28) A_B = It is possible to put a pseudo-hyphen into variables to make them more intelligible:

PLAYER_STRING = 10.

29) A$="His name is ""DAN"" and he's dead" = It is possible to use this sign('"')in a string by using it twice. In Atari Basic this would look like this:

A$="HIS NAME IS":A$(LEN(A$)+1)=CHR$(34):A$(LEN(A$)+1)="DAN":A$(LEN(A$)+1)=CHR$(34):A$(LEN(A$)+1)="AND HE'S DEAD".

Much better in "TB", isn't it ?

30) INKEY$ = Gives the last key pressed.

31) TIME = Gives the number of screen flips since timer reset. Setting it to zero resets the timer.

32) TIME$ = Gives the time in standard form. Has six signs and could be set this way:

TIME$="233500" - The time is 11:35:00 PM.

33) Pause A = Creates a delay equal to a screen flips (one screen flip is 1/50 of a second in Europe and 1/60 of a second in U.S.A.). The command actually gives a bigger delay because it takes a lot of time for basic to process it.

34) N=INSTR(A$,B$,A) = Searches the string in B$, in A$ starting from space No. A. If A is omitted then it searches from the beginning. N is the position in A$ where the B$ was found.

35) CLS,CLS #6 = Clears screen or graphics screen.

36) CIRCLE X,Y,R1,R2 = Draws an elipse using center position X,Y with the radius R1,R2. If R2 isn't specified, it will draw a circle with the radius R1.

37) PAINT X,Y = Fill area starting from coordinates X,Y (improved routine, and not the 'XIO 18,#6,0,0,"S:"' routine.)

38) FCOLOR = Color for paint

39) TEXT X,Y,A$ = Writes text on graphics screens with machine language or any other special knowledge. X,Y are positions according to graphics number.

40) DIR = Gives disk directory. DIR "P:" prints directory. DIR "D:*.???" gives directory of chosen files.

41) DELETE "D:FILE" = Deletes a file. (careful, no query).

42) RENAME "D:FILEOLD,FILENEW" = Renames filenames.

43) LOCK "D:FILE" = Locks a file.

44) UNLOCK "D:FILE" = Unlocks a file.

45) BRUN "D:FILE" = Runs a binary file.(Doesn't run everything)

46) BLOAD "D:FILE" = Loads a binary file.

=

BASIC3 TUTORIAL continued

47) BGET #1,LOCATION,X = Moves X bytes from file to 'Location'.

48) BPUT #1,LOACTION,X = Records X bytes from 'Location' to file.

49) GET A = Get value from keyboard. (No open command is necessary)

50) PUT A = Put A byte to screen.

51) CLOSE = Closes all channels (no numbers or other additives are necessary)

52) INPUT "NAME",A$ = This form is now legal.

53) SOUND = Same as close, but for sound - turns all sound channels off.

54) DSOUND A,B,C,D = Same parameters but enables double the time frequencies.

55) DSOUND = Same as sound.
56) DO ... LOOP = Creates an infinite loop which could be exited by the next command

57) EXIT = Exits the DO ... LOOP.

58) While A<B ... WEND = While A<B the loop will repeat itself, but when A)=B then the loop will continue to the first line after the 'WEND'.

59) REPEAT ... UNTIL A<B = Same as above.

The main differences between 58 and 59 is when the check is being performed - if A=B the 'WHILE' loop will stop at the beginning and will not perform when A=B but the 'REPEAT' will stop at the end and thus perform the loop while A=B.

60) IF = If has an enlarged mode and takes a few formats:

```
10 IF A=B      |
20  --PROGRAM  | 1
30  --PROGRAM  |
40 ENDIF

10 IF A=B      |
20  --PROGRAM  |
30 ELSE        | 2
40  --PROGRAM  |
50 ENDIF       |

10 IF A=B:----:ELSE:----:ENDIF  - 3
```

61) EXEC EXAMPLE = A procedure is a new kind of variable. It is a name of a subroutine in a "TB" program. It is much faster and more effective than GOSUB because it has a name which usually refers to its function, and it keeps the memory address of the line number and not the physical line. (The GOSUB command start to search the line from the beginnig of the program while the exec command jumps straight to the line)

62) PROC EXAMPLE = This way we start a procedure. At the end we have to have the command 'ENDPROC'.

This concludes this brief review of turbo-basic. Its little and requires experiments. Altough "TB" is a great program, it has some disabilities that I can discuss if someone is interested so for any questions, problems, comments, critics, etc. just drop me a line, and I promise to answer every letter.
:
My address is:

SAFERMAN OFER
21 BRANDE ST.
PETAH-TIQVA, 49600
ISRAEL

P.S. :The program this time is a game called invaders. It is very similar to the classic version and will show many useful uses of commands in TURBO-BASIC.

PRINTER INTERFACE.

Review by: RED.

When my old 1027 printer died, I didn't know what to do as I used it a lot for word processing. I decided to ask my English friends if they could help me out. One of my friends suggested that I contact TWAUG. I wrote to them and they sent me an address of someone they said might be able to help.

I wrote to this person telling him of my problem and he sent me a list of add-ons he made for the XL/XE computers such as the 256K and 1meg upgrades and a printer interface that fits inside the computer which allows you to run any Epson compatable printer on your 8-bit.

Well, this seemed to be just what I was looking for, and it was reasonably priced. If you are not good at soldering, then make sure you know someone who can help you. I was lucky in that way, my brother was able to help. There are one or two chips that have to be taken out and replaced with new ones which you can either plug in or solder and you also have to fit a socket on to the back of your computer for your printer cable.

When I received the package containing the interface, I was pleased to find that it had been assembled as far as was possible and came with very good fitting instructions. All the wires were connected to the chips, switch and to the 25 pin socket (all except one). All I had to do was, de-solder the old chips and replace them with the new ones. It was all very easy to fit with the very helpful instructions and diagrams.

Once everthing is fitted, connect up your Epson compatible printer, load your favourite word processor and give it a test. Don't worry if you have a few problems at first, when I did my first printout, the printer was going over the same line all the time as the paper wasn't moving up.

PRINTER INTERFACE continued

I checked up with my printer manual, altered the dip switches and all was well.

The big improvement for me now is that with my old 1027 printer I could only do text, but with my new printer and the very good printer interface, I can now also do very good screen dumps. I am now also able to use some of those other very good programs such as Print Shop and Daisy Dot and many others.

Although this is mainly a printer interface, it does however, have some other very useful features built-in. When you switch on the interface, you can choose either to have the internal Atari character set or the interface's own which is a larger font. You can also lock your keyboard which can be very useful when you have children, when you are busy working on the computer but have to leave it for a short time, you can lock the keyboard so that if someone were to press any of the keys, it would not damage your work. Another very useful option it has is that you can plug in your Atari tape deck, put on your favourite music tape and play it through your TV or monitor speaker while you work.

I have now fitted this interface to my 130XE without any help from my brother with the soldering. If you are thinking of changing from your Atari printer to an Epson compatible printer, then this is a very good and cheap interface to buy. The gentleman who makes these interfaces only makes them to order and they usually cost about £15 but this price can alter slighty depending on the changing price of chips.

I hope very soon to buy more add-ons for my computer and disk drive, so I will be writing a review on them for a future issue of the TWAUG newsletter.

SOFTWARE TIMERS

and how to use them.

by: Nir Darey

The ATARI O.S has 6 system softwere timers. (RTCLOCK,CDTMV1-5)

All of these system timers are being modified during VBLANK, RTCLOCK (18-20) or it's full name 'INTERNAL REALTIME CLOCK' is the first timer to be decreased when 'Immediate VBLANK" occurs, it doesn't count in seconds but 'JIFFIES', a jiffy is equal to 1/50 of a second in PAL compatible computers. Every Immediate VBI location 20 gets increased by 1 until it gets to 255, then at the next VBLANK register 20 gets reset to zero and location 19 gets increased, then after location 19 is equal to 255 it gets reset and location 18 increases and so on...

CDTMV1 (536,537) this is system timer one, every Immediate VBI the value at location 536,537 gets decreased by one and when it reached zero it does a JSR (Jump to SubRoutine) through the address written in location 550,551 (CDTMA1). The O.S uses system timer one for I/O routines, so I don't recommend this timer for your own use, unless you are writing your own O.S!

SOFTWARE TIMERS continued

The next four timers get updated during deferred VBI, if you remember what I wrote about the CRITIC register in the VBI article, you probably know that the computer checks the CRITIC register (66) after Immediate VBI occurres, if it's set (equal to 1) the O.S returns from the VBI and doesn't execute stage two of VBI (Deferred VBI).

CDTMV2 (538,539) this is system timer two, it's the same as system timer one with the difference that it's updated during deferred VBI, when the contents of location 538,539 reaches zero it does JSR through the address written in CDTMA2 (552,553).

CDTMV3 (540,541) the third system timer. when it reaches zero instead of JSRing through a vector it clears a flag at CDTMF3 (554). the cassette handler uses this timer to set the lengh of time to read or write tape headers. I won't recommend using this timer during cassette operation.

CDTMV4 (542,543) system timer four and CDTMV5 (544,545) system timer five are the same as system timer 3, when CDTMV4 reaches zero it clears flag at CDTMF4 (556), and when CDTMV5 reaches zero it clears flag at CDTMF5 (558).

Now let's look how to use the timers. To initialise a system timer counter to your own routine. It is done the same way as you initialise a VBI.
You can set the system counter address to point to your own routine, (for example 552,553 if you use CDTMA2). Another way is to load the Y register (LDY) with the low byte address of your routine, and the X register (LDX) with the high byte address of your routine, then load ACCUMULATOR with 1-5 for system timer 1-5 and do JSR 54860.
Another thing you must know, always at the end of your routine you must store another value at CDTMV register and do an RTS.

Now lets look at a small example:

```
10      *=1536
20      LDA #02
30      LDY #<CDTMA2
40      LDX #>CDTMA2
50      JSR 58460
60      LDA #1          ;for 1 jiffy
70      STA 538         ;count.
80      RTS
90 CDTMA2
100     LDA 20
110     STA 53274
120     LDA #1
130     STA 538
140     RTS
```

The source above was written in MAC/65.

Well that's all until next time.

CRACKING THE CODE

by Keith Mayhew

Re-printed by M. Gerum

This article first appeared in "The UK ATARI Computer Owners Club" later renamed "MONITOR"

Part 6

Last time we looked at the assembler and saw how it alleviated us from many monotonous tasks, the most simple of which are looking up opcodes for mnemonics, and values for labels - such as the location of a variable or the address of an instruction in the program. Also we do not have to bother ourselves with the fact that two-byte addresses are stored in reverse order and that branches use a relative offset: the assembler handles them all for us and does not make mistakes. However this does not mean that you should forget about the subtleties of how instructions are stored in memory and their effect on memory locations and registers when executed. Such an understanding will be found to be invaluable when developing programs, mainly beause it will give you confidence in what you are doing, but also, it will give you an in-depth view of what your program does when running; this will help most when you are trying to debug your code, especially at the machine code level of opcodes and operand bytes.

We have already studied each machine instruction in isolation in earlier articles, so you should now be in a position to write any program you like and you can even use an assembler to help you. Well, that is not the whole story: although we have seen each instruction and how to write it in assembly language form, we have not yet finished with the ways of addressing data in memory; without these vital addressing modes, which involve one of the index registers, virtually no real program could be written! Do not worry though, we shall look at these offending modes later and once they are understood, all you will lack is some experience. Of course experience will only come with practice, for which there is no substitute, but you will find that this will help you choose the appropriate set of instructions to perform a certain task. It is the ability to pick the optimum set of instructions that will make your program more efficient, that it will run faster, be smaller or use less memory space for its data; it is very difficult to find the best of these three, and in certain cases one may be more important than another. You need not concern yourself to deeply about optimising your program, the fact that it works will usually be of more importance to you!

A CASE OF EFFICENCY

Although your first attempt at a program may work, you may also find that it is large, slow and that trying to work out how the program actually does what it does could be something of a nightmare! Often clarity in a program is the most important item, this will be achieved by careful planning, whether it is on paper or in your head. A clearly coded program will need less comments to steer you through it when you are trying to change or debug it; remember that if you do not have a good approach to writing programs you will probably end up in such a tangled mess that debugging it will be nearly impossible. If your programs head this way then it is usually worth having a re-write before it becomes out of hand. Writing clear programs will only come with practice and you will develop your own style of writing, just as with any programming language; it is because of this that you may find the worst thing to do is try and understand large assembly listings of other authors, or maybe they write very messy programs anyway!

Writing a so-called efficient program can be roughly split into two areas: the first is selecting a good algorithm (method), this is important because it governs the way you will actually structure your program, and equally important to many programs is the way you structure any data you use in the program, these are very fundamental selections and have to be
taken before you can start writing your program. If you change your basic algorithm or the way your data is structured, then this will generally mean a large re-write of at least some of your program. The choice of these criteria usually has some relevance to the architecture, or internal layout of the processor - after all, it is pointless picking a very simple algorithm if it is going to be almost impossible to implement on the 6502 with any degree of efficiency.

Obviously the underlying algorithm and the structure, or layout, of your data will determine the maximum efficiency, but the second area where you can make a program faster and smaller,or some compromise between the two, will be in the actual selection of instructions to perform the job - in some cases, this means using some clever "tricks", but all this really means is understanding how instructions are executed and taking advantage of certain quirks, however, there are not many in the 6502 and most will be fairly obvious as and when you see them.

I have been purposely vague up to now because it is a matter of personal taste and experience which will determine exactly how you go about coding your program, and of course, the approach to two different programs will probably also be totally different. I will now mention just a few rough rules you might employ in your own programs.

FOLLOW THE RULES

The 6502 has a simple internal architecture consisting of only a few instructions and registers. The fact that the 6502 has a few instructions can actually be a blessing, after all, there is no vast array of complex instructions to remember and fortunately, there are no glaring ommissions which will handicap you severely; we shall see later that some of the real power of the 6502 lies with its addressing modes. Although the 6502 is easy to program because of its simplicity, its main limitations lie with the fact that there are only three general purpose registers at our disposal and there is some irregularity in the way they can be combined in different addressing modes (see tables in part 4). In fact, nearly all the common operations you need can only be performed on the accumulator, which means that in gereral a value has to be loaded into the accumulator, manipulated and then saved

CRACKING THE CODE continued

before the next operation can be performed. You will find that the X and Y registers are usually kept for use with the indexed addressing modes, but when they are free, they can be used to either save temporary values or they can be used, as they often are, as simply a counter. The key thing to remember is that the less memory fetches that are performed the faster the program will execute, that is, use the registers as fully as possible. A few operations can be performed directly on memory locations, without first having to load the value into a register, namely the increment, decrement and shift, rotate operations.

Special attention should always be made to the body of a loop. It is here that a group of instructions can be executed many times, thus any small time saving you can make in this section will improve the overall speed of the loop by a significant amount. It is very rare that you will be able to avoid memory accesses during a loop, so one way of saving time will be to use zero-page locations as this will save one or two cycles on every access (see any 6502 reference book for information on the number of cycles needed for each instruction/addressing mode). You will probably be aware that zero-page locations are quite valuable, especially when BASIC is around, so these have to be chosen carefully, but there use will help make a program faster and smaller.

Lastly, I will mention a few common 'tricks', these just make efficient use available instructions. Consider the following:

```
LDA NUM1
CLC
ADC  NUM2
STA  NUM1
LDA  NUM+1
ADC  NUM+2
STA  NUM+1
```

This is the common way of adding two, two-byte numbers together, which a compiler might generate for 'NUM1+NUM2' in some high-level language, where NUM1 is the low byte of the first number and NUM1+1 is the high byte of that number and similarly for NUM2. However, as we saw last time if the second number is a one byte number ie. NUM2+1 is equal to zero, then the NUM1+1 will either be unchanged or incremented by one, depending on the state of the carry. So, the last three instructions can be replaced by:

```
BCC NOINC   ;Increment
INC NUM1+1  ;If carry set.NOINC
```

This is less cumbersome than the first method and uses less bytes of storage for the code, also, for subtraction, a similar optimisation can also be made. If you only wish to increment the two-byte number by one, then the following could be used:

```
INC NUM1     ;Change low byte
BNE NOINC    ;Change high byte
INC NUM1+1   ;If low byte=zero. NOINC
```

In this example note that if the "INC" instruction does not change the carry flag, we have to decide when to change the high byte by a method which does not rely on the carry. It turns out that when the high byte needs incrementing when the low byte has changed from 'FF' to '00', so the high byte is changed when the low byte is equal to zero, hence the 'BNE' instruction.

The corresponding code for a decrememt, by one is slightly different, that is due to the fact we need to test for the low byte changing from '00' to 'FF', in this case we know that if the low byte is equal to zero before it is decremented then the high byte will also be decremented, so it can be coded as follows:

```
    LDA NUM1     ;Test low byte.
    BNE SKIP     ;Secrement high byte
    DEC NUM1+1   ;If low = zero.
SKIP DEC NUM1    ;Now do low byte.
```

Here the load needs to be performed to set the Z flag depending on the value of NUM1; It could of course, have used either the X or Y register instead. Just this one set of examples shows how you could write better code than you might have first used, and is the sort of optimisation a compiler normally cannot make, because it allows for the general case of an arbitrary two byte value to add to NUM1.

Other places where you could use similar tricks is in initialising memory values, for instance, if you wanted to set some locations to '01', some others to '00' and some others to 'FF' then you could use the following:

```
LDX #1      ;Inital value 1.
STX LOC1    ;Save.
STX LOC2    ;Save.
DEX         ;Decrement to 0.
STX LOC3    ;Save.
STX LOC4    ;Save.
DEX         ;Decrement to FF.
STX LOC5    ;Save.
STX LOC6    ;Save.
```

Fom this you can see that the use of one of the index registers is more useful than the accumulator as it can be incremented or decremented by one, in a single instruction. Apart from the more obvious fact that multiple stores of the same value should be done together (to save reloading the register), the useful feature is decrementing zero to obtain 'FF' and conversely, incrementing 'FF' to obtain zero. Note that 'FF' is the two's compliment of -1 for an eight bit number, so it makes sense to decrement zero to get -1 and incrementing -1 to get 0!

Another saving that is often made is that of the comparison of the value of counter to its end value, so instead of the following:

```
    LDX #0      ;Start at zero.
LOOP    .
        .       ;Body of loop.
        .
    INX         ; Increment count.
    CPX #$5B    ;Test for limit.
    BNE LOOP    ;Go back if not equal.
```

This is usually used:

```
    LDX #$5B    ;Start at maximum.
LOOP    .
        .       ;Body of loop.
        .
```

CRACKING THE CODE continued

```
DEC         ;Decrement count.
    BNE     ;Go back if not zero.
```

The comparison to zero is not needed at the end of the second version of the loop, because the decrement instruction will set the Z flag appropriately.

Many more examples can be found in the same vein, but it would take to much space to mention them all, however, I hope that it has got you to thinking that you can improve other code similarly. As a last point, remember that you can always use the stack to save temporary values on, by the "PHA" and "PLA" instructions to save and load the accumulator, but do not forget that you should always pull the same number of bytes that you push - otherwise you could be in serious trouble (unless you are up to some other devious trick!). It is therefore good practice not to place the 'PHA' and 'PLA' to far apart, otherwise you might forget which value was on the stack at that time. There are however, another two instructions which are rarely used, the 'PHP' and 'PLP' which perform a push and pull on the processor status flags register. You might use 'PHP', say after you have subtracted a number from the accumulator, this will save a copy of all the status flags, you could then continue working with the value in the accumulator and later use 'PLP' to retrieve the status from the subtraction, after which you would branch depending on the state of one of the flags, maybe the carry.

A solution to this sort of problem without using the above instructions could be quite messy. Another use of the push pull operations is to use 'PHP' followed by 'PLA' to get a copy of the status flags in the accumulator for testing and 'PHA' with 'PLP' to set the status flags to a certain state.

We will now return to the multiplication routine of last time to show how that can be improved.

IMPROVING MULTIPLICATION

If you recall, the routine in the last part of this series calculated the product of two 8-bit numbers by repeated addition to produce a 16-bit result. Just to show improvements can be made to programs, one astute reader pointed out, quite rightly, that there was little point in loading and saving the contents of the accumulator each time around the main loop. The improvement would be to take the load and store instructions out of the loop altogether, then when the loop had finished the accumulator would hold the low part of the result, so it could be saved back into RESULT at the point labeled EXIT and then follow it with the RTS instruction as before. Anyway, the algorithm used of repeated addition is hopelessly inefficent, especially so if the routine was extended to handle 16-bit numbers, due to the large number of iterations (repetitions) of the main loop which would be needed.

A better algorithm can be found by studying the way we normally multiply two decimal numbers together on paper, this involves considering the multiplicand as a whole and then considering the multiplication of each digit of the multiplier, in turn, with the multiplicand; the result is then the addition of these "partial products";

```
  225  Multiplicand.
  342  Multiplier.
 -----
  450
 9000  Partial products.
67500
------
76950  Result = sum of above.
```

The first partial product is 2*225 which is 450, the second is 4*225 but is shifted to the left by one digit ie. multiplied by 10, to give 9000, the third follows the same pattern, but is shifted left twice. If we apply this same method to binary numbers, then things simplify. Each digit of the multiplier will be either 1 or 0, so if we multiply by 0 the result will always be zero, if it is 1 then the result will just be the multiplicand. Each partial product will either be zero or the multiplicand itself, but rather than take a copy of the multiplicand and then shifting it left by the appropriate number of places, we can successively shift the multiplicand to the left by one each time around our loop. Listing 1 shows how we can code this algorithm.

The program starts by pulling the values of the multiplicand and multiplier off of the stack and storing them in MLTPND and MLTPLR. Next, the accumulator is loaded with zero to initialise the high byte of the multiplicand, location CChex, and both parts of the result. The X register is loaded with eight, which is the number of bits in the multiplier and hence the number of times we shall repeat the loop. The main loop starting at MULT, shifts the multiplier right so that the least significant bit is in the carry; if this is zero then we skip the addition of the multiplicand with the branch if carry clear instruction to SKIP. If the digit was a one, then the sixteen bit addition is made between the current value of the result and the multiplicand. Before the loop is repeated, we shift the two byte multiplicand to the left by one bit, this is achived by the ASL and ROL instructions, causing the carry to propagate between the two bytes and a zero to move into the least significant bit of the low byte, thus multiplying the multiplicand as a whole, by two. The bit count in the X register is then decremented and the loop continues if there is more to do. On each repetition of the loop, the multiplier will be successively shifted to the left thus effecting the multiplication by two each time.

Unlike the repeated addition method, this program will make exactly eight iterations of the main loop, whatever the two numbers to be multiplied together are. This saving is significant if you consider that if both programs were extended to handle sixteen bit numbers, the repeated addition method could make over sixty five thousand iterations of the main loop, this new method would need only sixteen.

We now have an efficent way of calculating the multiplication of two numbers, but again this to can be improved. If we call the multiplicand A and the multiplier B then the result will be A*B, now if we write B down in binary form, labeling each of the bits of B as b0 through to b7, we get:

B=b7*128+b6*64...b2*4+b1*2+b0

Thus for example, if B has the value 5 then b2 and b0 will be one and the rest will zero. We can now re-write this number as follows:

CRACKING THE CODE continued

B=bo+2(b1+2(b2+2(b3+2(b4+2(b5+2(b6+2*b7)))))

So the algorithm to work out the value of B goes like this: take bit 7 and multiply it by 2, add bit 6 to the result and then multiply the result by 2, then add bit 5 and multiply that result by 2 etc. This would continue until bit 0 had been added on in this case we would have the value of the binary number B. However we wanted the value of A*B so if you follow this through multiplying by A throughout, you simply end up with an A in front of each "b" in the last expression.

The algorithm to work out A*B is now: add bit 7 times A to the result and multiply it by 2, then add bit 6 times A and multiply that result by 2 etc., until we have added bit 0 times A. Of course each multiplication of a bit with A will only result in either zero or A itself, so is it really a case of considering whether to add A or not for each bit of B. Listing 2 shows how this algorithm can be implemented; note that this is not the complete listing - it omits the declaration of the three labels MLTPND, MLTPLR & RESULT and the start of the program which saves the values from BASIC in the first two variables, as in listing 1.

This routine uses the accumulator to hold the low part of the result and initialises this and the high byte of the result to zero at the start. Looking at the main loop we see that it first multiplies the result by 2, tests the next bit of the multiplier and then adds the multiplicand to the result if necessary. This is the same as the algorithm above, except that it multiplies the initial value of the result by 2, which of course, gives zero, this extra multiplication in fact saves instructions - so it is useful!. Listing 3 shows, yet another of doing this same multiplication which is even more efficent (and the last, I promise!). This algorithm can be deduced in a similar way to the above one; but I will not describe it fully here. This performs the multiplication by adding the multiplicand to the high byte of the result and then dividing the result by 2, using the ROR instructions, note here that two rotates are used, rather than shift and then a rotate because it also moves any carry from the previous addition down with it.

That's enough of multiplication routines! There are analogous methods for doing division - but they get worse, so I will not start embarking on such a boring pastime! I hope that after seeing four different ways of writing one program you may think about other ways of writing your own programs, or even re-writing other people's. Now we will start looking at those outstanding addressing modes.

ACCESSING DATA

So far, we have only been able to use absolute, or fixed, addressing to access data in the computer's memory, this is fine if all you want your computer to do is to keep multipling numbers together, but for any practical purposes we need a mechanism like that of the array in BASIC. For example, if we want to store a number in 256 consecutive locations, starting at some arbitary address then the only method we have so far is to use 256 store instructions!

We need a method whereby the processor sees a different address for the store instruction each time. A crude method of effecting this is by manipulating the actual operand bytes of the store opcode in the program, this is refered to as "modification" as we are modifying the actual code of the program; this can be useful in certain rare circumstances, but it is a method which you should try and avoid at all times! The problem with this method is that you will probably get very confused as to the current value stored for the operand, and unless it is reset, it will have a different value the next time the program is run - worse still is that the code could never be moved to a permament memory, or ROM, because it would be impossible to change the code, so the method wouldn't work! Having decided that modification is not the best method, we will now consider the methods that the 6502 offers us to access data.

INDEXING

To access a table of data which is at a fixed place in memory requires adding an offset to the basc address of the table. This mode of addressing is referred to as indexed, and not surprisingly, uses one of the two index registers; the assembler accepts the following to indicate this addressing mode;

```
STA  TABLE,X  ;Indexing with X register.
STA  TABLE,Y  ;Indexing with Y register.
```

The instruction is written as normal but has a comma and an index register name following the operand. If, for example, TABLE has the value of $4310, then the machine code will be the following: 9D 10 43 note that no extra information has been stored for this than an absolute access to location $4310, the only difference is the opcode of 9D instead of 8D for an absolute access. The opcode 9D still means store the accumulator but instructs the 6502 to use an index register in its access of the data; note that another opcode is used if we want to use the Y register, in this case it is 99 (see the opcode tables in Part 4). We will now see what happens when one of these new opcodes is executed; The operands are first fetched from memory, in this case it is $10 followed by $43, to form a sixteen bit address, now instead of using this address as it is ie. an absolute access, the contents of the specified index register is added on first to provide the final computed address which is used for the access, note that the index register is left unchanged. You can think of the instruction as been written as the following if you like:

```
STA  TABLE+X
```

However, this is not used by the assembler to indicate indexed addressing as it mght think you are writing a constant expression where X was a label name, so you will have to get used to reading the comma in the instruction as either 'indexed by' or 'plus'.

The following piece of code demonstrates the use of indexed addressing to store the number $5B into the 256 consecutive locations starting at $4310:

```
      LDA #$5B     ;data to store.
      LDX #0       ;Load index.
LOOP STA $4310,X ;Save using index.
      INX          ;Increment index.
      BNE LOOP     ;Loop until finished.
```

CRACKING THE CODE continued

Before the loop is entered, the accumulator is loaded with the data to be stored and the X register is loaded with the initial index of zero. When the store is first executed, the address will be computed to be $4310 as zero is being added on from X, so $5B is saved into location $4310. The value of X is then incremented by one and a branch is made back to the store, but this time the computed address will be $4311. This loop will continue storing the $5Bs into consecutive until the value of X returns to zero, hence all the locations from $4310 to $440F will be set to the value $5B by this indexing mechanism. Note that the Y register could have been used instead of X in the above program to exactly the same effect.

Now we have a way of accessing a table, we can write a small program to move a table from one address to another:

```
    LDX  #0       ;Zero index.
COPY LDA  TABLE1,X ;Get from first table.
    STA  TABLE2,X ;Place in second.
    INX           ;Increment common index.
    BNE  COPY     ;copy rest of table.
```

Assuming TABLE1 and TABLE2 are valid labels holding the address of the two labels, then the above program will read each consecutive byte of the first table and deposit the value in the second table using the accumulator to effect the transfer. Less than 256 bytes could be moved if we either compared X to a limit value before the branch, or by loading X with the limit value and then decremented X down towards zero.

The main limitation with the 6502's indexing is that as the X and Y registers are eight bits wide we can only access tables of 256 bytes or less in length. Another problem we are nearly always faced with is that the items or tables we wish to access keep moving around the memory! We will now see how the remaining addressing modes solve this problem.

INDIRECTING

The indirect addressing mode of the 6502 allows us to access data via a pointer in memory. For instance, if location $600 and $601 contain $10 and $43 respectively, then $600 and $601 are said to point 'indirectly' at location $4310, that is to say $600 and $601 hold the address of the location we wish to access. The 6502 only has one instruction which can actually use indirect addressing by itself - it is the jump instruction and is used like this:

```
JMP ($600)
```

Assuming that $CB and $CC still point to location $4310, then this instruction will jump to try and execute the code at location $4310. Again, no extra information is stored with the jump instruction, instead another opcode is used to unicate that the address specified is to be used indirectly, note that only the first address of the pointer is given; the second byte is always taken from the next consecutive address.

MIXING INDIRECTION WITH INDEXING.

All other instructions that can use indirect addressing eg. LDA or STA, cannot perform indirect addressing by itself, so the following is illegal to the assembler:

```
LDA ($600)
```

Instead of this, the 6502 limits us to page zero for the address of the pointer, and forces us to use indexed addressing with it, which is a combination of the two previous modes. The following show how we write this new mode into the assembler, note that in this case the X register cannot be substituted for Y:

```
LDA($CB,X)
```

This mode is termed as pre-indexed, as the contents of the X register is first added to the address $CB to give the final address (this will always be in page zero) from which the pointer is taken. So if X contains $10, then the 6502 will add this to $CB and go to the computed address of $DB; it will then take the contents of $DB and $DC as the pointer to the final location, which could be anywhere in memory. This mode is rarely used however, as it only allows us to access one item via a pointer, if we want to then access the next location in memory, we would have to do a two byte increment on the pointer in memory.

The last mode is called post-indexed and can only use the Y register, it is written as follows:

```
LDA($CB),Y
```

This post-indexed mode takes the contents of a page zero location and uses that indirectly, once it has obtained the location from this indirection, the Y register is added to the address to give the final address to be used. For instance, if $CB contained $10 and $CC contains $43 and Y contains $20, then $CB and $CC will be used to give the address of $4310 and the Y register will be added on to give the final address of $4330.

The use of this indirect-indexed mode allows us to have a series of pointers held in page zero which can then be used to access data tables in memory, indexed by the Y register, however, we still have the limitation that each table cannot be longer than 256 bytes, that is unless we change the value of the pointer. We shall explore the use of these modes in the next issue, but for now consider what Listing 4 does.

Listing 5 is a BASIC program which will allow you to load any object file you might have created with your assembler, but is mainly intended for cassette users as it will load the object files created by the Asembler/Editor cartridge. For cassette just type 'C:' in response to the prompt, disk users should type 'D:' followed by some file name as usual, however, you will probably find DOS more convenient unless you want to incorporate the routine in one of your programs.

Note that on line 10120 of the Basic program the '*' and the 'd' should be typed in inverse video. This is part of the machine code which speeds up the loading of the file considerably over that of using BASIC'S GET statement in a loop!

Now is your chance to get the assembler out again and do lot's of experimenting...

```
0100 ;Improved multiplication
0110 ;by partial products.
0120 MLTPND  =     $CB      ;Multiplicand.
0130 MLTPLR  =     $CD      ;Multiplier.
0140 RESULT  =     212      ;Result for BASIC.
0150         *=    $0600    ;Start on page 6.
0160         PLA           ;Discard number of data.
0170         PLA           ;Discard high byte.
0180         PLA           ;Get low byte.
0190         STA   MLTPND  ;Save as multiplicand.
0200         PLA           ;Discard high byte.
0210         PLA           ;Get low byte.
0220         STA   MLTPLR  ;Save as multiplier.
0230         LDA   #0      ;Y holds result low.
0240         STA   MLTPND+1;Zero multiplicand high.
0250         STA   RESULT  ;Zero result low
0260         STA   RESULT+1;and result high.
0270         LDX   #8      ;X holds bit count.
0280 MULT    LSR   MLTPLR  ;Get next bit of multiplier.
0290         BCC   SKIP    ;Skip if zero.
0300         LDA   RESULT  ;Get result low.
0310         CLC           ;and add
0320         ADC   MLTPND  ;multiplicand low.
0330         STA   RESULT  ;Save back.
0340         LDA   RESULT+1;Get result high.
0350         ADC   MLTPND+1;add multiplicand high.
0360         STA   RESULT+1;Save back.
0370 SKIP    ASL   MLTPND  ;Move the multiplicand
0380         ROL   MLTPND+1;one bit to the left.
0390         DEX           ;Decrement bit count.
0400         BNE   MULT    ;Loop if more to do...
0410         RTS           ;It's all yours BASIC!
```

Listing 1.

```
0230         LDA   #0      ;Zero result low (held in A)
0240         STA   RESULT+1;and high.
0250         LDX   #8      ;Load X with bit count.
0260 MULT    ASL   A       ;Multiply result by 2
0270         ROL   RESULT+1;and high byte.
0280         ASL   MLTPLR  ;Get next MSB bit in carry.
0290         BCC   SKIP    ;Skip if zero.
0300         CLC           ;Else add
0310         ADC   MLTPND  ;multiplicand.
0320         BCC   SKIP    ;Skip if no carry to high.
0330         INC   RESULT+1;Else add one to high.
0340 SKIP    DEX           ;Decrement bit count.
0350         BNE   MULT    ;Loop back if more to do.
0360         STA   RESULT  ;Save low byte of result.
0370         RTS           ;Finished.
```

Listing 2.

```
0230         LDA   #0      ;Zero high byte (in A)
0240         STA   RESULT  ;and low byte of result.
0250         LDX   #8      ;Load bit count.
0260 MULT    LSR   MLTPLR  ;Get next bit of multiplier
0270         BCC   SKIP    ;Skip if zero.
0280         CLC           ;Else add
0290         ADC   MLTPND  ;Multiplicand.
0300 SKIP    ROR   A       ;Divide result by 2
0310         ROR   RESULT  ;and low byte.
0320         DEX           ;Decrement bit count
0330         BNE   MULT    ;and loop if more.
0340         STA   RESULT+1;Save high byte of result.
0350         RTS           ;and say bye-bye!
```

Listing 3.

```
0100 ;Call from BASIC with
0110 ;X=USR(1536).
0120         *=    $0600
0130         PLA
0140         LDY   #0
0150 LOOP    TYA
0160         STA   ($58),Y
0170         INY
0180         BNE   LOOP
0190         RTS
```

Listing 4.

```
XQ 10 DIM FILE$(14)
DA 20 CH=1:? "Please enter device/file na
   me..."
AQ 30 INPUT FILE$
MC 40 GOSUB 10000
YT 50 END
RU 10000 OPEN #CH,4,0,FILE$:TRAP 10160
JY 10010 GET #CH,X:GET #CH,Y
WJ 10020 IF X<>255 OR Y<>255 THEN 10170
QU 10030 GET #CH,SL:GET #CH,SH
NQ 10040 ST=SL+256*SH
VA 10050 GET #CH,EL:GET #CH,EH
TU 10060 EN=EL+256*EH
BA 10070 IOCB=832+CH*16:LN=EN-ST+1
LR 10080 LH=INT(LN/256):LL=LN-LH*256
JO 10090 POKE IOCB+2,7
MC 10100 POKE IOCB+4,SL:POKE IOCB+5,SH
FN 10110 POKE IOCB+8,LL:POKE IOCB+9,LH
JE 10120 X=USR(ADR("hhh▸LVg"),CH*16)
QX 10130 GET #CH,SL:GET #CH,SH
YW 10140 IF SL=255 AND SH=255 THEN 10130
YL 10150 GOTO 10040
WA 10160 CLOSE #CH:RETURN
TW 10170 ? "File does not have binary hea
   der.":RETURN
```

Listing 5.

MARK'S GAMES COLUMN

by Mark Stinson.

This issue we have solutions to two adventures under the Adventure International flag, Earthquake - San Francisco 1906, and Waxworks.

Both adventures have been solved quite comprehensively by Flintstone (!) and comprise the solution, list of articles/uses, and maps.

The response to my appeals for hints and tips has been very poor, and I am most grateful to the few contributors who have submitted recently. I am now very short of contributions so why not have a go? If I receive your hints or tips, or full solutions they will be used, so get tipping!

As there is only a limited number of commercial adventures for us Atarians which have not already been reviewed, I will be looking at some good PD adventures in coming issues. If you have written an adventure which you would like me to look at then why not send a copy to TWAUG, and you may see your game getting the thumbs up in this mag.

SOLUTION TO EARTHQUAKE

LOOK-LOOK DRESSER-LOOK ENVELOPE-GET LETTER-READ LETTER-GET WRD OF BILLS-S-MOVE BED-GET CROWBAR-LOOK-LOOK-LOOK-LOOK-MOVE BEAM-WITH CROWBAR-LOOK-LOOK WALLS-OPEN DOOR-LOOK-TALK-PAY OWNER-GET APPLE-S-LOOK-CLIMB-DIG RUBBLE-GET GOLD WATCH-CLIMB DOWN-E-LOOK-CLIMB-LOOK-DIG MASONRY-LOOK-W-N-N-GET HANDGUN-CLIMB-LOOK-TALK-BRIBE SOLDIER-CLIMB-DROP APPLE-DROP HANDGUN-S-LISTEN-CLIMB-LOOK-GET SMALL CHILD-CLIMB DOWN-LOOK-GET GOLD KEY-S-LOOK-LOOK-TALK-LOOK-LOOK-LOOK LUMBER-GET LUMBER-N-N-DROP LUMBER-W-W-OPEN DOOR-S-E-LOOK-UNLOCK BOX-OPEN BOX-DROP GOLD KEY-GET SILVER KEY-W-N-N-GET HAMMER-CLIMB-CLIMB-GET HANDGUN-UNLOCK GATE-OPEN GATE-E-LOOK STREET-LOOK MANHOLE-LIFT COVER-DROP LETTER-S-LOOK UP-LOOK LIGHT-CLIMB-LOOK-KILL SOLDIER-WITH HANDGUN-LOOK SOLDIER-OPEN PACK-DROP HAMMER-CLIMB-N-CLIMB-W-UNLOCK GATE-OPEN GATE-DROP HANDGUN-W-W-OPEN DOOR-LOOK FRUITSTAND-GET SMALL DOG-N-CLIMB-CLIMB-GET APPLE-GET LUMBER-UNLOCK GATE-OPEN GATE-PULL GATE-DROP SILVER KEY-GET IRON POLE-E-LIFT COVER-S-CLIMB-DROP SMALL DOG-DROP APPLE-GET HAMMER-GET IRON NAILS-S-E-E-MAKE LADDER-DROP HAMMER-W-W-N-GET APPLE-GET SMALL DOG-S-E-E-GET LADDER-CLIMB LADDER-DROP LADDER-S-DROP SMALL DOG-LOOK-JUMP-LOOK-CLIMB HORSE-RIDE HORSE-LOOK PRECIPICE-LOOK CREVICE-LOOK QUARTZ-LOOK INDENTATION-LOOK FLAT SPOT-LOOK OBJECT-GET DIAMOND-E-LOOK-W-N-JUMP-GET SMALL DOG-GET PADDLE-N-DROP SMALL DOG-LOOK-S-GET SMALL DOG-N-OPEN DOOR-S-LOOK-SIT DOWN-EAT-DROP PADDLE-GET FORTUNE-READ FORTUNE-DROP FORTUNE-GET PADDLE-N-E-LOOK-VAULT-LOOK-LOOK PAGODA-GET GLASS-DROP DIAMOND-CLIMB-CLIMB-VAULT-W-LOOK CHINATOWN-LOOK CHINATOWN-TALK-GET BRASS KEY-E-VAULT-DROP IRON POLE-CLIMB-UNLOCK DOOR-OPEN DOOR-DROP BRASS KEY-S-LOOK-UNDRESS WOMAN-GET GREEN DRESS-E-DROP SMALL DOG-LOOK-LOOK-LOOK-W-N-GET BRASS KEY-UNLOCK DOOR-OPEN DOOR-UNLOCK DOOR-OPEN DOOR-DROP BRASS KEY-S-E-LOOK-CLIMB-LOOK-LOOK DOORWAY-PRY OFF BOARDS-DROP WAD OF BILLS-DROP PADDLE-DROP GREEN DRESS-W-LOOK-DROP CROWBAR-CLIMB-E-LISTEN-LOOK SIGN-OPEN DOOR-GET WAD OF BILLS-GET PADDLE-GET GREEN DRESS-OPEN DOOR-CLIMB-LOOK-WEAR GREEN DRESS-LOOK-S-CLIMB-LOOK-PADDLE-PADDLE-PADDLE-SWIM-SWIM-SWIM-CLIMB-PADDLE-PADDLE-PADDLE-PADDLE-LOOK-CLIMB-LOOK-LOOK-LOOK-JUMP-S-TALK-PAY HAMPTON

SMALL CHILD	GETS GOLD KEY
WAD OF BILLS	PAYS HAMPTON
CROWBAR	WOOD BEAM, MANHOLE, BOARDS
APPLE	HORSE

SMALL DOG	DIGS FOR PADDLE & HOLE AT COBBLESTONE STREET
SILVER KEY	OPENS GATE
WATCH	BRIBE FOR SOLDIER
HANDGUN	SHOOT SOLDIER AT FIRE STATION
HAMMER	
MAILS	MAKES LADDER
BOARDS	
GOLD KEY	OPENS IRON BOX
IRON POLE	VAULT OVER CRACK
BRASS KEY	OPENS DOOR IN PAGODA
PADDLE	ROWS BOAT & WOOD SCRAP
DIAMOND	CUTS GLASS
GREEN DRESS	FOOLS SOLDIER AT DOOR

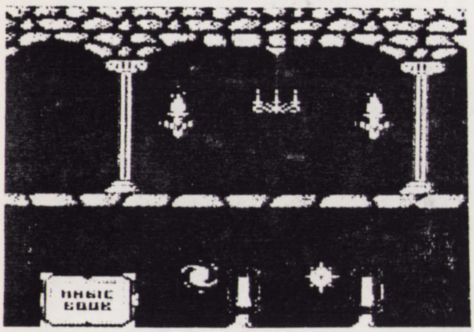
SOLUTION TO WAXWORKS

INVENT-READ PAPER-DROP PAPER-EXAMINE SEATS-INSERT COIN-U-W-SAY 46-E-W-SAY 1993-S-D-N-EXAMINE TOILETS-GET JACKET-EXAMINE JACKET-EXAMINE JACKET-DROP JACKET-GET PISTOL-GET BULLETS-LOAD PISTOL-S-W-S-W-S-GET AQUALUNG-WEAR AQUALUNG-S-W-GO AIRLOCK-SWIM-W-S-N-EXAMINE JUNK-EXAMINE JUNK-GET LAMP-GET CROWBAR-RUB LAMP-D-EXAMINE PIED PIPER-MOVE GUY FAWKES-DROP PISTOL-DROP CROWBAR-GET FLUTE-U-OPEN CUPBOARD-U-D-GET BEAM-DROP AQUALUNG-GET KEY-N-GO GRID-LIGHT TORCH-E-E-W-W-DROP KEY-GET TRAP-EXAMINE TRAP-P-DROP TRAP-GET CHEESE-E-E-PLAY FLUTE-RUB LAMP-DROP BEAM-PLAY FLUTE-D-DROP CHEESE-U-GET KEY-D-DROP KEY-U-U-D-GET AQUALUNG-WEAR AQUALUNG-RUB LAMP-D-W-W-S-W-G AIRLOCK-SWIM-W-D-RUB LAMP-GET BEAM-RUB LAMP-FASTEN BEAM-RUB LAMP-D-DROP AQUALUNG-GET PISTOL-GET CHEESE-PLAY FLUTE-WAIT-DROP MATCHES-GET ROPE-TIE ROPE-GET MATCHES-PLAY FLUTE-GO WELL-PLAY FLUTE-GO TUNNEL-DROP CHEESE-EXAMINE ALTAR-GET TALISMAN-WEAR TALISMAN-DROP FLUTE-N-GO DOOR-EXAMINE JEWELLERY-GET CASKET-RUB LAMP-D-GET KEY-OPEN CASKET-DROP CASKET-DROP KEY-GET BARREL-WAIT-GO WELL-E-SHOOT ZOMBIE-DROP PISTOL-GO DOOR-STRIKE MATCH-LIGHT FUSE-DROP BARREL-RUB LAMP-D-GET CROWBAR-GET TANNA-WAIT-GO WELL-E-GO DOOR-GO PASSAGE-OPEN SARCOPHACUS-GIVE TANNA-WEAR MASK

KEY	OPENS GRID IN WASHROOM
TELEPHONE	RED HERRING
COIN	INSERT IN SLOT MACHINE TO WIN TORCH
WOODEN BEAM	SECURES TRAPDOOR
AQUALUNG	TO GO AIRLOCK
DISPLAYS	ANSWER QUESTIONS TO GET AQUALUNG
JACKET	CONTAINS PISTOL & BULLETS
PISTOL & BULLETS	KILLS ZOMBIE
RAT TRAP	GIVES CHEESE
CHEESE & RATS	GNAWS ROPE AT ALTAR
LAMP	TRANSPORTER
CROWBAR	OPENS SARCOPHACUS
ROPE	TO GET DOWN WELL
GOLDEN KEY	OPENS GOLDEN CASKET
TALISMAN	PROTECTS FROM MUMMY
APPLE	RED HERRING
FLUTE	MAKES RATS FOLLOW
BARREL	HOLDS GUNPOWDER CLEAR PASSAGEWAY
MATCHES	LIGHTS FUSE IN BARREL
MUMMY	GIVES GOLDEN MASK
GOLDEN MASK	WEAR TO FINISH GAME

IMAGINE

Reviewed by Mark Fenwick.



From Poland comes "Imagine", an animated adventure with quite a few tricks up it's sleeve. The copy Dave sent me (cheers Dave!) is a playable demo version. The finished game will be released through Micro Discount, hopefully in time for the AMS7 show.

The story, well there has to be, Wormthop (nice name, must be Polish ?) successfully locates the ruins of the Mikki Temple containing the legendary Secret Force Crystal. After removing it and returning to his cave he hopes to discover if it's powers were true. After many experiments he falls asleep, not realising the Crystal has begun it's work. Wormthop awakens to find himself in the strange place of The Haunted Castle of Kirthrop, as described by the ancients. This place is known to be the most dangerous place to be, the only way back to the normal world is to pass the two graves and coffins in the death room. So it's up to you, can you save Wormthop ? or is he to be banished forever...

On boot up, we see the title screen accompanied by a lively piece of music, underneath the title are the credits to the author. Pressing fire here takes us to the game play.

The main playing area takes up about two thirds of the screen, with the lower third showing your status. To the lower left is a magic book to help you with your tasks ie. Shoot Fog, Shoot Fire, Wall Key, Make Door plus more. To the right of this are two Davy Lamps the left represents supply of Fog, while the other shows supply of Fire. A burning candle at the far right shows amount of health. Each time you touch a nasty, the candle will burn a little lower. Once the candle burns out completely you sink in to the floor and die, a gravestone complete with cross and the letters R.I.P. marks the spot of your death.

The play area consists of the stone floors, walls and ceilings of the Castle. There are variety of things featured in the scenery, chandeliers, staircases, burning torches and ladders to name a few. The view is taken from the side, to give a feeling of depth. Graphics are well detailed no blocky graphics here, no corners cut, even on our main character, who's well detailed with a cloak, shoes, white beard and hair.

As for playability, very smooth, the movement of character very realistic, almost life like. Choose weapons, and prompts from the book by pulling back on joystick and pressing button to select. Left, right, up, down and diagonal for climbing or descending stairs. No jumping involved in this game, just leisurely strolling around. There are various nasties to contend with on your journey, bats are easily killed while others such as fire breathing gargoyles and what looks like a sleepwalker, may need a little more effort. There's plenty of screens to work your way through,

for each level a key is needed to open doors to progress further. While playing this demo version I counted more than twenty screens with a lot more I didn't see, so there's some depth for you!

Overall I think this game is excellent, great music, top notch graphics, colour and gameplay. Everything of an adventure but without the boredom of the run of the mill N,S,E and W. Well done you lads in Poland, you're just the people to prove that the 8-Bit is far from dead! "Imagine" will soon be available from Micro Discount at a quite moderate price of £6.95.

APOLOGY FROM MAX

Max sends his apologies for not being able to include his Textpro article in this issue. ON the 19th of October, he received a letter from the Freeman Hospital in Newdastle making arrangements for him to go in to have his five way heart bypass operation. Max goes into hospital on Monday the 25th of October ready to have the operation on the following day.

Max is very disappointed not to be going to the show this year, he was looking forward to meeting some of you there. He hopes that he will be recovered enough to be able to do his article for issue seven but we have told him to take it easy until he is fully recovered.

I know that all of you would like to Join John and myself in wishing Max a speedy recovery.

David Ewens.

AAARGH!

SHOCK! HORROR!
we've been criticized!

Someone has written to us voicing concern over our £12 life membership scheme...because it's too cheap!?

Membership of the Atari Classic Programmer's Club entitles those who are determined to get the most out of their amazing machines to assistance with absolutely any aspect of programming the Atari Classic.

So why not join up with a life membershop...before we change our minds! For more details, send a large SAE to, ACPC, Pen-Tyddyn, Capel Coch, Llangefni, Anglesey, Gwynedd, LL77 7UR, Wales.

DISK CONTENT

Programmes on side A all have DOCS on the disk.
FAST BASIC V2.0: A very fast Basic Interpreter.
INDEX CREATOR: Sorts and prints out alphabetically.
TITLECARD: Make your own title screens.
SPOT THE DIFFERENCE: Make up your own screens.
There are two games on side B.
SPY HOTEL: Kill the spies before they get you.
THE GLASNOST GAME: This is a Tetris clone.

Enjoy

COMING SOON...

MENU PRINT ELITE

version 1.0

Remove all those disk cataloguing hassles with MENU PRINT ELITE, the complete Atari 8-bit disk cataloguing system. Features include:

- Ability to read a large number of menu and DOS systems

  including Rob C, Multiboot, DOS 2.0, DOS 2.5, DOS 4, DOS XL, SpartaDOS, Howfen Menu, Howfen DOS, Transdisk and others..
- Create and print labels.
- Assign individual disk numbers for easy storage of disks.
- Additional user input feature for cataloguing of unrecognisable disks.
- Dump catalogue to disk for printing out later.
- And much, much more ...

So, if you've got a large collection of disks but don't know what's on them, Menu Print Elite will find out for you.

Release Date: November 1993 Price £9.95

ACPC

The Atari Classic Programmer's Club.

MICRO DISCOUNT

Here is some good news for all 8-bitters, MICRO DISCOUNT (Derek Fern) will release two New Hardware kits and twelve New Software Titles at the A.M.S.7 show.

NEW HARDWARE

STEREO CONVERSION KITS

NEW SOUND REPLAY CARTRIDGE

NEW SOFTWARE

" THE BRUNDLES "

STEREO DRUM EDITOR "8 CHANNEL DRUM MACHINE"

DRACONUS and ZYBEX both in "STEREO SOUND"

16 BIT SOUND PLAYER DISK

OPERATION BLOOD "LIGHT GUN VERSION"

IMAGINE BANK BANG

SPECIAL FORCES WHEEL OF FORTUNE

TANKS THE MERCENARY

ARTIFACT